UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

IMPLEMENTAÇÃO DE UM ALGORITMO DE BUSCA COM CONTROLE DE TRÁFEGO DISTRIBUÍDO E ADAPTAÇÃO DE TOPOLOGIA PARA REDES PEER-TO-PEER BASEADAS NA PLATAFORMA JXTA

por

ANDRÉ PANISSON

Projeto de Diplomação

Prof. Lisandro Zambenedetti Granville
Orientador

Porto Alegre, agosto de 2003

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitora: Profa. Wrana Panizzi
Pró-Reitor Adjunto de Graduação: Professor Norberto Hoppen
Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux
Coordenador da Comissão de Graduação do Curso: Prof. João César Netto
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

Agradecimentos

Agradeço a todos os colegas e aos professores do Instituto de Informática que nestes quatro anos e meio de convivência contribuíram decisivamente para que se este objetivo fosse concretizado.

Ao prof. Lisandro Zanbenedetti Granville pelo incentivo e pela paciência como orientador.

À minha querida noiva, Anelise, pelo companheirismo, força e compreensão. Igualmente aos meus pais Alfeu e Inês e meus irmãos Renato, César, Jonas e Gelson.

Aos meus amigos, Alex, Roger e Élton pela paciência em me agüentarem como colega de apartamento.

Finalmente, ao pessoal da CIC/99 pelas muitas tardes de estudo e companheirismo nesses anos de graduação.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Na última década, a Internet cresceu de maneira a suportar uma infinidade de aplicações. Novas aplicações surgiram e ganharam espaço entre os usuários, e entre elas obtiveram destaque as aplicações *peer-to-peer* (P2P), que permitem a qualquer usuário, denominado *peer*, compartilhar recursos com outros usuários na Internet. Apesar de ter surgido a poucos anos, o compartilhamento de recursos através de sistemas P2P representa hoje uma grande fração do tráfego na Internet.

Dentre as características das redes P2P, uma das principais é a escalabilidade. Porém, é provado que muitas das tecnologias P2P utilizadas não são totalmente escaláveis e à prova de falhas. O principal empecilho para a escalabilidade dessas aplicações é o seu algoritmo de busca por recursos na rede, cujo comportamento causa um crescimento exponencial no número de mensagens enviadas. Esse tipo de algoritmo de busca de recursos também possui outros problemas de comportamento, como o efeito curto-circuito.

Além desse algoritmo, que é amplamente utilizado nos sistemas P2P atuais, existem outras soluções apresentadas por diversos grupos de pesquisa, a maioria delas envolvendo algoritmos que se utilizam de DHTs – *dynamic hash tables*. Porém, esses algoritmos são ótimos para busca de recursos com um identificador conhecido, mas não são bons para buscas aproximadas.

Este trabalho apresenta um algoritmo alternativo por caminhamento proposto por Lv, Ratnasamy e Shenker [LV 02], escalável, descentralizado e baseado em controle de fluxo e adaptação da topologia entre os vários *peers* da rede. Este algoritmo também é uma opção capaz de tirar o máximo proveito das características de heterogeneidade da rede Internet.

Aqui propõe-se também a implementação desse algoritmo sobre a plataforma JXTA, uma plataforma atualmente em desenvolvimento pela comunidade da Internet, e com a grande vantagem de possuir código aberto. Entre os seis protocolos da plataforma JXTA, um deles – o *RendezVous Protocol* – é reimplementado, de forma a ter o comportamento descrito pelo algoritmo acima mencionado.

A fim de verificar o comportamento atual da plataforma e compará-lo com o comportamento da nova implementação, é feita também a implementação de uma aplicação capaz de testar as funcionalidades necessárias a uma típica aplicação P2P. É feita então uma análise do protocolo implementado, os problemas existentes na nova implementação, os problemas que surgiram e como foram resolvidos, os caminhos tomados e o porque de cada decisão de implementação, e os trabalhos futuros sobre o assunto a serem desenvolvidos.

Abstract

In the last decade the Internet has grown in a way it allows many uses. New applications have came and gone, but some have fixed their places among Internet users, and the peer-to-peer (P2P) application is one of them; the one which allows any user – called *peer* – to share files with other Internet users. Although it is a brand new application, the sharing of files through P2P systems is responsible today for a huge part of the Internet traffic.

One of the main features of P2P networks is scalability. But it has been proved that many of the P2P tecnologies are not completely scalable, nor failure proof. The main problem in the scalability of these applications is their network searching algorithm, whose behavior may cause a exponential growth in the number of sent messages. These type of searching algorithm has yet other behavior problems, like the short-circuit effetc.

Besides this algorithm, which is largely used in today's P2P systems, there has been other solutions presented by many researching groups, most of them using algorithms that use DHTs – dynamic hash tables. Although these algorythms are excelent for file searching with exact match query, they are not good to not exact match searches.

The present work presents an alternative scalable algorythm using walking, as presented by Lv, Ratnasamy and Shenker [LV 02], de-centralizer and based on flow control and topology adaptation between the many network peers. This algorithm is also the option that takes the best out of the Internet's heterogenity feature.

In the present work we also propose an implementation based on the JXTA platform to this algorythm, which is nowadays being developed by the Internet community, taking advantage of its open code. Among the six JXTA platform's protocol, one of them – the Rendevous Protocol – is re-implemented in a way that follows the above-mentioned algorythm.

In order to verify the behavior of today's platform and compare it to the behavior of the implemented one, we also make an upgrade to one application in order to test the functionality of a typical P2P application. It is then made an analysis of the implemented protocol, of the troubles that arise and how they were solved, of the paths taken in order to solve these problems and the justificative of each decision taken in the implementation, and an overview of future works on this subject.

1. Introdução

Com o crescimento da Internet nos últimos anos, muitas novas aplicações surgiram e ocuparam espaço entre os usuários, e entre elas obtiveram destaque as aplicações *peer-to-peer* (P2P). O crescimento no uso dessas aplicações demonstra que a arquitetura P2P surge como uma nova forma de tecnologia, não substituta à arquitetura cliente-servidor, mas tão importante quanto esta última.

Junto a este crescimento no uso das aplicações P2P, houve também um violento aumento na quantidade de tráfego, devido em boa parte aos algoritmos pouco otimizados de busca utilizados pelas aplicações. Uma aplicação P2P deveria ocupar toda a capacidade do usuário no que é realmente produtivo, que é o compartilhamento de recursos entre os *peers*, e não gastar boa parte de sua capacidade em um de seus serviços, como é o caso da capacidade usada na busca por recursos. É preciso portanto estudar maneiras de otimizar este serviço, de forma a minimizar a quantidade de recursos gastos com ele. Porém é preciso manter a escalabilidade da rede formada por esses *peers*, evitando qualquer comportamento centralizado e não escalável.

Na primeira parte do trabalho, é dada uma visão geral sobra a arquitetura P2P, sua origem desde o programa Napster, sua história, suas características, e como se diferencia da arquitetura cliente-servidor. A arquitetura P2P não deve ser considerada como uma arquitetura substituta à arquitetura cliente-servidor, mas pode ser considerada tão importante quanto esta última. Muitas das funcionalidades por definição existentes em uma aplicação P2P, como *download* simultâneo e inerentemente escalável de milhares de arquivos, não seriam possíveis em uma aplicação cliente-servidor tradicional, pelo fato de que a distribuição de recursos em uma rede P2P ser algo que faz parte de sua natureza. Esse capítulo trata dessas questões, e lista algumas das principais diferenças entre a arquitetura P2P e a arquitetura cliente-servidor, com as vantagens e desvantagens de cada uma delas.

A busca pelos recursos existentes na rede é a primeira função importante a ser desempenhada por uma boa arquitetura P2P. Na segunda parte do trabalho, discutem-se esses algoritmos de busca, quais os mais utilizados, seus principais problemas e o porque de serem considerados não-escaláveis. Em substituição a esses não escaláveis, existem outros atualmente em estudo, particularmente os algoritmos baseados em DHT – *dynamic hash tables*. Esses últimos são ótimos para busca de recursos que podem ser descritos por um identificador único previamente conhecido. Porém não são tão eficientes para o caso de buscas aproximadas. Este capítulo apresenta o algoritmo proposto por Lv, Ratnasamy e Shenker [LV 02], baseado em controle de fluxo e adaptação de topologia. Esse algoritmo cria uma rede auto-estruturada, de maneira a formar uma hierarquia que não é explícita, mas que é montada de forma que os *peers* com mais capacidade de transmissão fiquem no topo da rede e sejam os primeiros a receberem os anúncios de recursos, e os primeiros a serem consultados ao ser feita uma busca.

A terceira parte descreve a plataforma JXTA, uma plataforma de código aberto disponível a desenvolvedores de aplicações P2P. O projeto JXTA define um conjunto de protocolos para o desenvolvimento de aplicações P2P, a fim de resolver o problema de incompatibilidade entre os protocolos do grande número de sistemas P2P existentes. O principal objetivo do projeto JXTA é definir uma rede P2P genérica que possa ser usada para a implementação de uma grande variedade de aplicações e serviços P2P. Este capítulo descreve os principais conceitos na plataforma JXTA: *peers*, *peergroups* e *advertisements* ou anúncios. Descreve todos os seis protocolos pertencentes à plataforma, e o objetivo e funcionamento de cada um deles. Para

terminar, faz um breve comentário sobre o atual algoritmo de busca, que é definido pelo *RendezVous Protocol*, e sobre o que a comunidade de desenvolvedores da plataforma JXTA pretende modificar neste protocolo em um futuro próximo.

O capítulo seguinte descreve a implementação de uma aplicação sobre a plataforma JXTA. É uma aplicação feita para testar as funcionalidades da rede a ser montada, e verificar o seu comportamento, a fim de compará-lo com a nova implementação feita. Descreve a parte cliente da aplicação, a parte servidora e os anúncios que são trocados entre os diversos *peers*. Faz também a análise do comportamento da aplicação quando conectada à rede JXTA pública, que já possui centenas de *peers* conectados e pode ser considerada uma rede de grande escala, e faz também a análise da aplicação quando conectada a uma rede privada, com apenas alguns *peers*, mas cujas conexões podem ser facilmente verificadas.

O último capítulo descreve a implementação feita do *RendezVous Protocol*, de forma que o comportamento do *peer* reflita o algoritmo anteriormente apresentado. Descreve as principais classes usadas, os métodos de cada classe, e os trechos de código que são essenciais para o funcionamento da plataforma, além de trechos de código que podem ser modificados de forma a alcançar pequenas modificações de comportamento que, através de uma análise mais detalhada, podem até mesmo aumentar o desempenho da aplicação. Este capítulo também faz um apanhado geral de trabalhos futuros que podem ser feitos sobre esse mesmo assunto.

2. A arquitetura P2P

Na última década, a Internet cresceu de maneira a suportar uma infinidade de aplicações, ricas tanto em variedade quanto em tamanho. As últimas a serem adicionadas a este conjunto foram as aplicações de compartilhamento *Peer-to-Peer*, ou P2P, que permitem a qualquer usuário, denominado *peer*, compartilhar arquivos e outros recursos com outros usuários na Internet. Apesar de ter apenas poucos anos de uso, o compartilhamento de recursos através de sistemas P2P representa hoje uma grande fração de tráfego na Internet, em algumas situações até mesmo acima do tráfego Web.

2.1. Origem do P2P

O fenômeno P2P começou em 1999 com o lançamento do programa Napster [FRA 02]. No mês de janeiro, Shawn Fanning, um estudante de Ciência da Computação da Universidade Northeastern, criou esse programa, que permitia a usuários compartilharem arquivos de áudio (MP3). Em maio, foi criada a empresa Napster, Inc. Em agosto, o *site* Napster.com iniciou o seu serviço e se tornou a sensação momento.

Em uma rede Napster, usuários (ou *peers*) guardavam seus arquivos MP3 em disco local, enquanto o sistema rodava um servidor central guardando apenas um índice contendo os arquivos disponíveis à comunidade de usuários. Para fazer o *download* um determinado arquivo, o usuário fazia uma pesquisa no servidor central baseada em palavras-chave, e obtinha assim o endereço IP de todos os *peers* que possuíam os arquivos com as palavras-chave usadas na pesquisa. O usuário podia então efetuar o *download* dos arquivos desejados de um destes *peers*.

A idéia por trás do programa Napster é muito simples – o armazenamento distribuído de arquivos nos usuários (ao invés de servidores) acompanhado por um índice centralizado a fim de facilitar a localização desses arquivos. Essa idéia simples provou ter grande sucesso. Apesar disso, provou ter um ponto vulnerável: a centralização do índice de arquivos. Vulnerabilidade provocada pelo fato de que, além de o sistema não ser totalmente escalável, possuir um único ponto de falha. Em 7 de dezembro de 1999, foi iniciada uma batalha judicial comandada pela Associação das Gravadoras dos EUA, a *Recording Industry Association of America* (RIAA), reclamando de violações de direitos autorais pela facilitação de compartilhamento de arquivos MP3, que só terminou em março de 2001, como desligamento do servidor de indexação. Sendo o servidor de indexação crucial para o funcionamento do sistema, este servidor central foi obviamente o principal alvo. Acabando com o sistema de indexação central seria uma maneira fácil de terminar com todo o sistema.

Naquela mesma metade de 1999, enquanto Shawn Fanning estava ocupado facilitando o compartilhamento de arquivos MP3 entre computadores, um outro cientista da computação de Londres travalhava em uma estratégia para assegurar a liberdade de expressão na Internet. Seu nome era lan Clark que, no mês de junho, completou o estágio inicial da arquitetura conhecida hoje como FreeNet [CLA 01]. Embora sendo, assim como o Napster, um programa feito para compartilhamento livre de arquivos, o FreeNet difere do Napster em alguns aspectos importantes, e pertence a uma classe de serviços conhecidos como Sistemas de Banco de Dados Distribuídos com Criptografia, ou *cryptographically distributed databases* (CDDs).

Seguindo o crescimento do Napster e FreeNet, sistemas similares como Gnutella [RIP 01], Jungle Monkey [CAR 98], MojoNation e outros surgiram

rapidamente. Esses sistemas seguiram o modelo de descentralização de armazenamento do sistema Napster, porém eram diferentes na maneira de encontrar os arquivos. Ao invés de um índice central em um servidor, esses sistemas utilizam técnicas de busca descentralizadas, onde uma pesquisa é propagada entre os *peers* e qualquer *peer* que possuir um arquivo que corresponda à pesquisa responde diretamente ao *peer* requerente. Essa nova geração de sistemas de compartilhamento de arquivos é completamente descentralizada tanto em termos de armazenamento quanto em termos de busca, uma mudança motivada principalmente pelos motivos legais e técnicos acima mencionados. Para escapar dessas limitações, esses sistemas foram projetados de forma que não houvesse um único ponto de centralização. Em março de 2001, o Napster foi terminado. Porém isto não significou o fim dos sistemas de compartilhamento de arquivos. Pelo contrário, os sistemas completamente descentralizados continuam, e novos sistemas descentralizados como Kazaa e Emule surgiram, fora o fato de que a população de usuários P2P continua crescendo dia a dia.

2.2. Crescimento das redes P2P

Além dos novos sistemas descentralizados que estão surgindo, deve-se considerar também o fato de que a população de usuários P2P continua crescendo dia a dia. Este crescimento pode ser observado, conforme Subharata Sen [SEN 02] na análise do tráfego dos roteadores de borda de um grande provedor de Internet, durante três meses, a partir de outubro de 2001. Este estudo revelou que um volume total de aproximadamente 1.2 Tera-Bytes por dia pode ser atribuído a não mais que três dos sistemas P2P mais populares – FastTrack, Gnutella e DirectConnect. Além disso, o mesmo relatório citado anteriormente [FRA 02] estima que entre 400 e 600 mil arquivos de filmes são trocados diariamente na Internet usando estes sistemas, com uma quantidade total de mais de 9 milhões de usuários simultâneos.

2.3. Importância das aplicações P2P

Apesar do desenvolvimento dessas aplicações remeter a um conjunto de questões legais, sociais e que interessam diretamente a toda comunidade da Internet, principalmente ao se falar em controle de tráfego, de uma perspectiva técnica o mais importante aspecto do P2P são suas vantagens tecnológicas. Para verificar estas vantagens, podemos fazer uma comparação, considerando se estas aplicações fossem desenvolvidas utilizando a tecnologia existente da Web. Seria a arquitetura P2P uma solução nova realmente necessária para distribuição de conteúdo?

2.4. Web X P2P

Deste o início da década de 90, a Web serviu como um meio efetivo de publicação e acesso a conteúdo na Internet. Com sua arquitetura cliente-servidor, os documentos são guardados em servidores. Para acessar um determinado documento, um usuário, ou cliente, localiza o servidor, envia uma requisição e obtém uma resposta. Tipicamente, um documento está guardado em um número reduzido de servidores, e existe um número muito menor de servidores do que de clientes. Além disso, esses servidores tendem a estar *online* por períodos relativamente longos de tempo (da ordem de dias) enquanto clientes tendem a estar menos tempo acessíveis. Por exemplo, a duração de uma conexão discada pode ser da ordem de minutos. Documentos na Web são nomeados utilizando *Uniform Resource Locators* (URLs). Por exemplo, a URL

`http://www.inf.ufrgs.br/exemplo.htm` representa o arquivo `exemplo.htm` localizado no servidor `www.inf.ufrgs.br` através do protocolo `HTTP`. O meio de busca dos servidores é o *Domain Name System* (DNS).

Por mais de uma década, a Web serviu para disponibilizar conteúdo a milhões de usuários. Porém a arquitetura de sistemas P2P é bastante diferente da arquitetura cliente-servidor. Em sistemas P2P, um *peer* age como cliente e como servidor. Os *peers* buscam, guardam e servem conteúdo. Diferentemente de servidores Web, os *peers* são altamente transitórios, entrando e saindo do sistema em períodos curtos de tempo. Enfim, diferentemente da Web, um sistema P2P não possui uma única e definida maneira de nomear o conteúdo, como URLs. Os usuários são livres para nomear os arquivos guardados em sua máquina local da maneira que desejarem. As duas arquiteturas possuem maneiras diferentes de distribuição de conteúdo:

- A arquitetura P2P permite que o conteúdo seja publicado fácil e rapidamente. Na Web, é necessário primeiramente obter um domínio DNS e configurar um servidor DNS para que resolva este novo domínio. A propagação de informação pela adição ou remoção de nomes de domínio através de sistemas DNS é um processo relativamente demorado, e normalmente requer configuração manual. Em sistemas P2P, a publicação de conteúdo resume-se à cópia dos arquivos desejados a um diretório em sua máquina local.
- A principal maneira de procurar conteúdo, tanto na Web quanto em sistemas P2P, é por busca através de palavras-chave. Esse sistema é efetivado na Web através de servidores de busca centralizados, como Google e Yahoo, que constroem índices centralizados de conteúdo através do processo de *crawling* na Web. Esse processo é inteiramente independente da vontade do usuário. Embora novos documentos na Web possam ser acessados conhecendo a exata URL, o mesmo documento não estará disponível em *sites* de procura enquanto este não fizer o seu processo de *crawling* sobre esse documento. Tal solução é apropriada quando o conteúdo fica no sistema por períodos relativamente longos de tempo, porém não é apropriada para documentos que são freqüentemente adicionados e removidos. Na arquitetura P2P, por outro lado, uma vez que um *peer* entra no sistema, os arquivos publicados estarão imediatamente disponíveis tanto para busca como para *download*.
- Diferentemente da Web, os sistemas P2P se baseiam inteiramente na participação das máquinas dos usuários, e não necessitam de nenhuma infra-estrutura preexistente. Na Web, recursos de armazenamento e de rede são normalmente disponibilizados por servidores de grande capacidade. Além disso, seu funcionamento depende da infra-estrutura DNS. Isto torna seu funcionamento independente do comportamento do cliente. Os sistemas P2P, por outro lado, não fazem uso de infra-estrutura preexistente, e os recursos disponíveis aos usuários são a soma dos recursos disponíveis em cada máquina de usuário existente na rede.

Por esses e outros motivos, o fenômeno de distribuição de aplicações P2P não poderia ter acontecido sobre uma infra-estrutura similar à Web. A atração dos sistemas P2P está em seu potencial em permitir o desenvolvimento rápido e de baixo custo de aplicações poderosas e de grande escala. Porém, as promessas da arquitetura P2P só poderão ser realizadas se os sistemas forem realmente escaláveis.

3. O problema da busca em redes P2P

Há duas funções importantes que sistema P2P deve desempenhar: a busca para encontrar a informação desejada e o *download* efetivo da informação. O armazenamento descentralizado nos sistemas P2P faz do processo de *download* inerentemente escalável. A parte difícil é como encontrar o *peer* de onde trazer a informação, e fazer isso de forma escalável.

Soluções de pesquisa nos sistemas atuais caem em duas categorias: Os sistemas centralizados, como o Napster; e descentralizados como Gnutella, Kazaa, FreeNet e outros.

Soluções centralizadas já foram citadas como vulneráveis devido ao seu ponto de falha, além de ser de difícil escalabilidade quando trata-se de sistemas da ordem de milhões de usuários. Porém, serviços de procura como Yahoo e Google demonstraram que é possível construir tais soluções. Isso, porém, implica em investimento significante em infra-estrutura em servidores e alta largura de banda. Além do fato de que a alta volatilidade dos *peers* e a constante entrada e saída de informações no sistema torna o modelo de busca centralizada completamente diferente dos atualmente existentes, pois não poderá ser utilizado o processo de *crawling*.

Esses problemas levaram à adoção de soluções descentralizadas. Essas soluções envolvem tipicamente a organização dos *peers* em uma *overlay network*, sobre a qual as requisições de procura são passadas de *peer* para *peer*. Funcionando dessa maneira, os sistemas atuais criam novos e diferentes problemas de escalabilidade. Por exemplo, no Gnutella, as mensagens de busca sofrem um processo de *flooding* dentro de um determinado escopo de *peers*. O processo de *flooding* em cada requisição certamente não é escalável – o número de mensagens circulando na rede cresce exponencialmente. Há também o fato de o processo de *flooding* ter de ser cortado em um determinado ponto, o que determina seu escopo, podendo assim falhar em encontrar o arquivo procurado em alguns sistemas grandes. FreeNet, um outro sistema de busca descentralizado, possui um algoritmo de busca baseado em caminhamento randômico, que pode falhar em encontrar arquivos mesmo que atualmente estejam presentes no sistema.

3.1 O efeito Curto-Circuito

O excesso de mensagens causado pelo comportamento do *flooding* não é o único problema em uma rede que usa esse tipo de algoritmo. Um outro problema bastante interessante descrito por Mihajlo A. Jovanovic [MIH 01] é o problema do efeito curto-circuito. Esse efeito pode ser descrito pela análise do comportamento de uma rede funcionando com algoritmo de *flooding* que tenha características de heterogeneidade nos atrasos de transmissão, como é o caso da Internet.

O *flooding* nos sistemas P2P atuais funciona através de controle de *loops* e *time-to-live* (TTL). Para o controle de *loops,* cada *peer* guarda em *cache* os identificadores das mensagens recebidas recentemente. Se o *peer* receber uma mensagem que foi recentemente enviada a ele, a mensagem é automaticamente descartada. Isso evita que uma mensagem ocupe desnecessariamente recursos de rede. Para o controle via TTL, cada mensagem enviada possui um campo denominado *time-to-live*, que é decrementado a cada vez que a mensagem é re-enviada. Quando esse campo chegar a zero, a mensagem é descartada.

Teoricamente, se ao campo TTL de cada mensagem for atribuído o diâmetro da rede (isto é, o maior caminho mínimo entre dois *peers*), todos os *peers* da rede receberão uma cópia da mensagem. Na prática, porém, quando a rede possui diferentes atrasos de transmissão entre os pontos, a mensagem pode não chegar a uma quantidade significativa de pontos.

Por exemplo, considerando os *peers* P1, P2, P3, P4 e P5, com as conexões descritas na figura 3.1. O diâmetro da rede é igual a 3 (o maior caminho mínimo é entre P2 e P5 ou entre P1 e P5). As conexões representadas por linhas mais finas são conexões com atraso da ordem de um segundo, enquanto as conexões representadas pelas linhas mais largas são conexões com atraso da ordem de um milésimo de segundo.

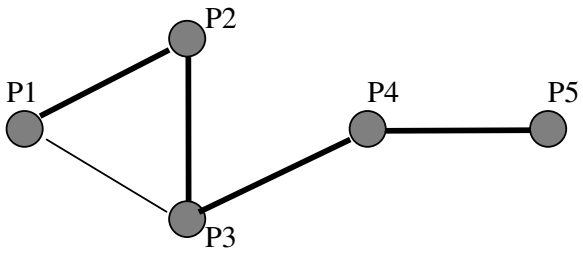


Figura 3.1. Exemplo de rede com diferentes atrasos de transmissão

Supondo que uma mensagem seja enviada a partir do *peer* P1, com TTL igual a 3. Essa mensagem passa pelos peers P2 e P3, e chega rapidamente ao *peer* P4, sendo ali descartada (TTL = 0). Algum tempo depois, a mensagem chega ao *peer* P3, e é descartada devido ao controle de *loops* (a mensagem já passou por P3). Nesse caso, a mensagem jamais chegará ao *peer* P5.

O problema é muito mais freqüente em uma rede de grande escala. Ele mostra como o algoritmo atualmente usado pelos sistemas P2P utilizando *flooding* pode ter problemas de desempenho e comportamento, mesmo em redes pequenas. Segundo Mihajlo A. Jovanovic, em seu estudo na rede Gnutella, a redução no número de *peers* alcançados chega a uma média de 55%, uma parte considerável de *peers* na rede.

3.2. Algumas soluções encontradas para o problema

Motivados pelos problemas de escalabilidade nos sistemas P2P, muitos grupos de pesquisa investigam esta questão: pode ser desenvolvida uma solução escalável para o problema de localização de recursos em redes P2P? Como resultado, um número de grupos de pesquisa tem proposto sistemas que podem ser chamados de "altamente estruturados". Um exemplo desses sistemas estruturados é descrito por Helder [HEL 02], e pode ser considerado quase que um algoritmo de roteamento, além de ter grandes aplicações em comunicação *multicast*. Nesses sistemas estruturados, a topologia da rede é controlada, e os arquivos (ou referências aos arquivos) são colocadas em lugares específicos. Esses sistemas estruturados fornecem um mapeamento entre os identificadores de arquivos e sua localização, de forma que as buscas sejam eficientemente roteadas até os *peers* com os arquivos desejados (ou então até uma referência ao arquivo desejado). Esses sistemas, desta forma, oferecem uma solução realmente escalável para pesquisas de busca. Porém é preciso considerar que estas buscas apenas poderão ser roteadas ao *peer* responsável se for possível fazer uma busca exata, em que o identificador completo do objeto especificado é conhecido. Essas buscas são as chamadas *exact match query*, que diferem consideravelmente das

pesquisas feitas por aproximação, bastante comuns em sistemas de bancos de dados modernos.

3.3. *Exact Match* e Sistemas DHT

Entretanto, esses sistemas estruturados são caracterizados por serem menos flexíveis em um ambiente de uma população de usuários muito transitória, especialmente porque há um custo alto para manter esta estrutura (como lista de vizinhos, etc.) requerida para que o roteamento funcione eficientemente quando os *peers* tem uma taxa alta de entrada e saída do sistema. Além disso, está demonstrado que estes sistemas, apesar de funcionarem bem para buscas exatas, não são tão eficientes em técnicas de busca aproximada, tais como busca por *substring*, o que é comum em sistemas atuais de compartilhamento de arquivos.

Os exemplos desses sistemas estruturados utilizam algoritmos DHT – *Distributed Hash Table Systems* – cuja funcionalidade está provando ser muito útil, inclusive para grandes sistemas distribuídos, e um grande número de projetos está propondo a construção de facilidades na Inernet feitas sobre algoritmos DHT. Alguns exemplos são sistemas de arquivos globais como PAST [DRU 01], CFS [DAB 01], e Ocean Store [RHE 03] que utilizam DHTs para localização dos arquivos e dados. Os algoritmos mais conhecidos de roteamento baseado em DHTs são: Tapestry [ZHA 01], Pastry [ROW 01], Chord [STO 01] e CAN (*Content Adressable Networks*) [RAT 02]. Porém, todos esses sistemas são capazes de fazer buscas apenas por chaves exatas, as chamadas *exact match query*.

3.2. Busca aproximada

Apesar da extensa pesquisa sobre algoritmos baseados em DHTs e em busca exata, existem poucos relacionados à busca aproximada. Um exemplo básico de busca aproximada é a busca por arquivos feita atualmente nos programas de compartilhamento de arquivos. Por exemplo, digamos que um determinado usuário compartilhe um arquivo chamado `ArquiteturasP2P.pdf`. a busca por um arquivo cujo nome contém a *substring* `P2P` deve encontrar o arquivo acima compartilhado. Não é possível fazer esta busca através dos algoritmos que se utilizam de DHTs, pelo fato de que não se pode saber, de antemão, o nome exato do arquivo a ser localizado. Sabendo que não é possível utilizar nenhum dos algoritmos acima mencionados, é possível desenvolver um sistema escalável que faça essa busca?

Sistemas como Gnutella já se preocuparam com esta questão, e desenvolveram algumas soluções que remediam o problema do *flooding*, mas não o resolvem totalmente. A solução adotada pelo Gnutella é a construção de redes baseadas em *SuperPeers*. Os *peers* de usuário se conectam aos *superpeers* e estes guardam uma tabela com todos os arquivos compartilhados pelos usuários que estão conectados a eles. É feita assim uma estruturação da rede de forma que os *peers* com maior capacidade de transferência, transformados em *superpeers*, possuem muitas conexões com os *peers* menores, e, pelo fato de possuírem cópias das descrições dos arquivos compartilhados pelos usuários, não precisam enviar o *flood* aos peers de usuário, mas o *flood* é feito somente entre os *superpeers*. Desta forma, o número de mensagens geradas por uma busca se reduz bastante.

O problema na utilização de *superpeers* é que o algoritmo de *flooding* limitado por TTL ainda é utilizado, e enquanto este for o algoritmo básico de busca, não pode-se considerar essa solução como escalável.

Considerando todas estas questões, os pesquisadores Qin Lv, Sylvia Ratnasamy e Scott Shenker [LV 02] propõem um algoritmo de busca que pode resolver estes problemas. Esta solução não utiliza *flooding* e sim um caminhamento randômico, onde cada nodo, ao invés de enviar a pesquisa a todos os seus vizinhos, escolhe randomicamente um vizinho e envia a pesquisa a ele. Melhor ainda: o algoritmo aproveita muito bem as características de heterogeneidade reveladas por redes P2P.

3.3. Projeto do algoritmo

Esse projeto para uma rede P2P não estruturada usa um algoritmo de controle de tráfego distribuído e um algoritmo de construção de topologia que:

- Restringe o tráfego de consultas em um determinado *peer* de modo que ele não se torne sobrecarregado;
- Faz com que a topologia da rede evolua dinamicamente, de modo que as mensagens sejam redirecionadas aos nodos que possuem capacidade suficiente para tratá-las.

Sobre essa topologia de rede baseada na capacidade dos nodos, utiliza-se um caminhamento randômico com pesos, em que os nodos com maior capacidade tem mais probabilidade de receber mensagens. A combinação destes algoritmos resulta em um sistema cujas mensagens são rapidamente direcionadas a nodos com maior capacidade, além de, pela utilização de uma técnica de replicação de objetos, haver maior probabilidade de encontrar as respostas desejadas. Esse sistema resulta em uma organização semi-hierárquica de nodos com alta capacidade nos níveis mais altos da hierarquia. Essa hierarquia, porém, não é explícita; ela é sim adquirida através de um algoritmo distribuído, adaptativo e auto-organizável.

3.3.1. Controle de tráfego e adaptação da topologia

No que segue, representa-se a capacidade de um nodo i por C_i. Neste algoritmo assume-se que C_i representa o número máximo de mensagens que um nodo i pode processar em um dato intervalo de tempo T. Um nodo i é conectado a um conjunto de nodos vizinhos, representados por $viz(i)$. Para cada $j \in viz(i)$ o nodo i mantém as seguintes informações:

- $in[j,i]$: o número de mensagens que chegaram do nodo j para o nodo i no último intervalo de tempo T. Cada nodo i informa sua taxa total de mensagens chegadas: $in[*,i] = \sum_{j \in viz(i)} in[j,i]$

- $out[i, j]$: o número de mensagens que o nodo i enviou ao nodo j no último intervalo de tempo T.

- $outMax[i, j]$: o número máximo de mensagens que o nodo i pode enviar ao nodo j por intervalo de tempo T.

Quando um novo nodo i entra na rede, ele é conectado a um conjunto randômico de nodos. Para cada novo vizinho j, i inicializa:

- $outMax[i, j] = \frac{C_i}{D_i}$
- $in[j,i] = 0$
- $out[i, j] = 0$

Cada nodo *i* conta o número de mensagens que recebe e transmite a cada vizinho j. Periodicamente o nodo *i* verifica se está sobrecarregado (a taxa de recebimento de mensagens excede sua capacidade). Se sobrecarregado, o nodo *i* tenta adaptar a topologia da rede de forma a reduzir sua taxa de recebimento – o nodo *i* seleciona um nodo *p* com alta taxa de envio (alto $in[p,i]$) e redireciona a outro vizinho *q* com sobra na taxa de recebimento. A sobra na taxa de recebimento do nodo *q* é calculada por $C_q - in[*,q]$. Intuitivamente, a regra de adaptação de topologia anterior cria uma conexão direta entre um nodo com alta taxa de envio de mensagens (*p*) e um nodo com alta capacidade (*q*) e tira do caminho o nodo sobrecarregado. Se o nodo sobrecarregado *i* não puder encontrar um nodo *q* apropriado (isto é, todos os seus vizinhos estão perto de sua capacidade máxima), o nodo *i* envia um pedido ao nodo *p* para que reduza o número de mensagens enviadas ao nodo *i*.

Nota-se que todas as decisões sobre controle de tráfego e ajuste de topologia são baseadas em informações locais (referente ao nodo e a seus vizinhos somente). Além disso, se a capacidade do nodo sofrer uma rápida modificação ou se o nodo sair do sistema, a topologia se adaptará automaticamente a fim de acomodar as mudanças.

3.3.2 Algoritmo de busca

O sistema usa uma busca baseada em caminhamento randômico que combina características de diversos outros algoritmos. Utiliza-se TTL a fim de terminar o caminhamento, e guarda-se o estado da busca pelos nodos onde passou, a fim de acelerá-la. O nodo passa as mensagens adiante aos seus nodos com maiores valores em *outMax*. Isso fará com que as buscas rapidamente cheguem aos nodos com maior capacidade.

Em resumo, um nodo *i* que recebe uma mensagem, repassa a seu vizinho *j* com maior valor em $outMax[i,j]$ entre todos os $j \in viz(i)$ de forma que $out[i,j] < outMax[i,j]$ e *i* não enviou a mesma mensagem ao nodo *j*.

A fim de aumentar o desempenho do algoritmo, utiliza-se algumas estratégias de replicação. Quando um nodo quiser oferecer um recurso a ser compartilhado, este recurso – ou uma referência a ele – será replicada em diversos outros nodos. Assume-se que estas cópias serão distribuídas randomicamente entre os nodos, porém a probabilidade de essa replicação ser feita em um dado nodo será proporcional à sua capacidade. Não necessariamente assume-se assim que um nodo com alta capacidade de transferência deverá ter também uma capacidade de armazenamento maior que a dos outros nodos, pois este espaço de armazenamento não deverá ser tão grande.

Este é o algoritmo a que este trabalho se propõe implementar e discutir seus problemas e algumas melhorias, utilizando a API de programação da plataforma JXTA, que é o próximo assunto.

4. A plataforma JXTA

O projeto JXTA [GON 01] – pronuncia-se juxta, de *juxtapose* – é um projeto de código aberto concebido inicialmente pela Sun Microsystens, Inc. e depois posto ao encargo de um grupo independente, sob a licença denominada *Apache Software License*. Foi desenvolvido com a participação crescente de instituições acadêmicas e da indústria de software. O projeto JXTA define um conjunto de protocolos para o desenvolvimento de aplicações P2P, a fim de resolver o problema de incompatibilidade entre os protocolos do grande número de sistemas P2P existentes. O principal objetivo do projeto JXTA é definir uma rede P2P genérica que possa ser usada para a implementação de uma grande variedade de aplicações e serviços P2P. Define também um conjunto de blocos a serem reutilizados e um conjunto padrão de regras a serem seguidas. Os desenvolvedores estão livres para substituir qualquer um destes blocos criando seus próprios blocos e criando suas próprias regras, porém respeitando as regras padrão. Por exemplo, a plataforma pode ser customizada a fim de usar novos algoritmos de roteamento ou busca.

4.1. Conceitos importantes na plataforma JXTA

A plataforma JXTA assume que a rede P2P seja totalmente adaptativa, de modo que as conexões podem ser feitas e desfeitas a qualquer momento e os *peers* possam entrar e sair da rede no momento que desejarem. Os caminhos físicos podem até mesmo ser unidirecionais (é o caso de *peers* dentro de redes com *firewalls* que só permitem conexões a partir de um endereço interno da rede), e os caminhos de roteamento podem mudar tão rapidamente quanto a topologia da rede muda.

4.1.1. *Peers* e *PeerGroups*

Os *peers* da rede JXTA se organizam em grupos denominados *peergroups*. Um *peergroup* representa um conjunto de *peers* que tem interesses comuns, e concordam com um conjunto comum de regras. Os protocolos JXTA descrevem como esses grupos serão criados, publicados e descobertos, e como o *peer* pode descobrir como deverá se comportar. Esses grupos formam regiões lógicas na rede que não necessariamente refletem sua topologia física. Um *peer* pode fazer parte de quantos grupos desejar, e criar quantos grupos achar necessário.

A plataforma JXTA foi projetada especialmente para atender *peers* que podem ser qualquer tipo de dispositivo. A especificação dos protocolos determina expressamente que os *peers* da rede devem ser assumidos como sendo qualquer tipo de dispositivo, desde os menores aparelhos com tecnologia embarcada até o maior *cluster* de supercomputadores. Desta forma tentam eliminar barreiras à participação de quaisquer sistemas operacionais, plataformas ou linguagens de programação. É assumido também que os *peers* e seus recursos podem aparecer e desaparecer espontaneamente da rede e a localização de um *peer* pode mudar espontaneamente ou até mesmo ser mascarada por equipamento de NAT ou *firewall*.

4.1.2. *Advertisements*

Todo e qualquer recurso da rede JXTA é representado por um anúncio, ou *advertisement*. Esses anúncios são representados como metadados independentes de linguagem, através de documentos XML. Através de anúncios, o problema de encontrar *peers* e todos os seus diferentes tipos de recursos se reduz ao problema de encontrar

anúncios que descrevem estes recursos. Os *peers* guardam, publicam e trocam anúncios a fim de descobrir e alocar os recursos disponíveis. Os recursos só podem ser descobertos através de uma pesquisa de seus anúncios. Todos os anúncios são publicados com um tempo de vida e, após este tempo, o anúncio expira e se torna inválido.

4.2. Protocolos JXTA

Uma solução P2P completa oferece ao *peer* mecanismos para:

- Descobrir outros *peers*, seus serviços e recursos;
- Publicar seus serviços e recursos disponíveis;
- Trocar informações com outro *peer*;
- Rotear mensagens até outros *peers*;
- Verificar as informações disponíveis em um determinado *peer*;
- Agrupar os *peers* em grupos

A plataforma JXTA define um conjunto de protocolos projetados para as funcionalidades comuns requeridas, a fim de que a rede P2P seja formada independente do sistema operacional, linguagem de desenvolvimento e rede de transporte existente em cada *peer*.

A especificação dos protocolos faz também algumas recomendações importantes. Uma dessas recomendações é de que os *peers* guardem informações em *cache* a fim de reduzir o tráfego em rede. Outra recomendação é de que se preveja roteamento de mensagens para os *peers* que não estiverem diretamente conectados ou acessíveis à rede (*firewall traversing)*.

Os protocolos JXTA estabelecem uma rede virtual sobre a rede física existente. Essa rede virtual fornece primitivas simples para esconder a complexidade da topologia da rede física, permitindo que qualquer *peer* se comunique com qualquer outro *peer* da rede. Cada recurso da rede é identificado unicamente através de um ID e endereçado, independentemente de sua localização, através de uma relação entre o seu endereço lógico e o endereço físico desse recurso. As mensagens são roteadas de forma transparente, e podem até mesmo atravessar *firewalls* e NATs, inclusive utilizando diferentes protocolos a fim de chegar a seu destino. Os protocolos padronizam a maneira como é feita a descoberta dos *peers*. Estes podem se organizar em grupos, publicar e descobrir recursos, comunicar-se e monitorar um ao outro.

Baseado nos critérios anteriores de implementação e outros ainda existentes na especificação dos protocolos (*Protocols Specification* [JXTA 03]), o grupo JXTA desenvolveu um conjunto de seis protocolos baseados em mensagens XML, como mostrado na figura 4.2.

O *Endpoint Routing Protocol* (ERP) fornece um mecanismo para determinar rotas até um determinado *peer*, permitindo que os *peers* se comuniquem utilizando camadas de transporte incompatíveis. Por exemplo, dois *peers* diferentes podem conectar-se à rede JXTA, um através de TCP, e outro através de HTTP (note-se que o protocolo HTTP está sendo usado como camada de transporte) e trocar mensagens de forma transparente. Permite também que mensagens sejam enviadas através de *firewalls* e NATs. Todas as camadas superiores ao ERP fazem uso de abstrações chamadas *endpoints*. Os *endpoints* mascaram o tipo de camada de transporte que está sendo usada, permitindo assim que um *peer* envie e receba dados independentemente da camada de transporte. A figura 4.1 ilustra a forma como é feita a comunicação entre dois *peers* protegidos por *firewall* através do ERP.

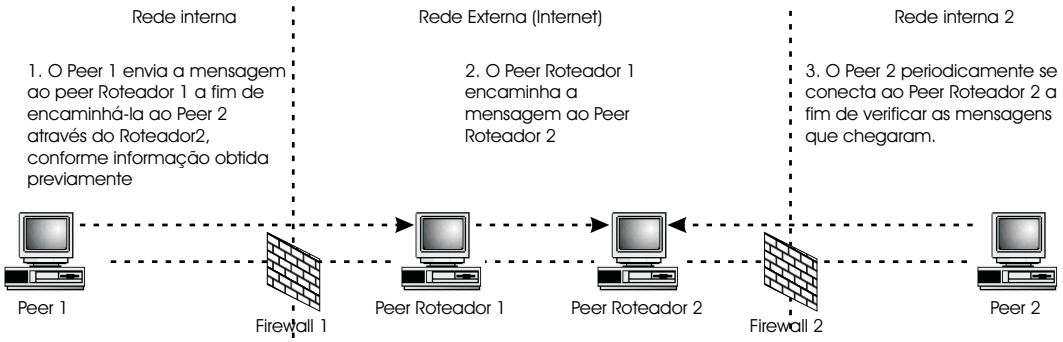


Figura 4.1. Mensagem ERP enviada do *Peer* 1 ao *Peer* 2, atravessando 2 *firewalls*

O *Peer Resolver Protocol* (PRP) trata do envio e recebimento de mensagens através de *queries*. Define um protocolo para o envio de uma determinada *query* a um destino localizado em outro *peer*, e para o processamento de uma *query* recebida.

Pipes são construções que enviam ou recebem dados de um *peer* remoto. Os serviços da plataforma JXTA utilizam tipicamente o PRP ou um *pipe* a fim de comunicar-se com outro *peer*. Antes de um *pipe* ser usado, é necessário que ele esteja ligado a um *peer* remoto. Os *pipes*, como tudo na plataforma JXTA, também são representados por anúncios. O *Pipe Binding Protocol* (PDP) define um conjunto de mensagens – *queries* e *responses* – que um *peer* pode usar a fim de relacionar um *pipe* a um *peer* remoto.

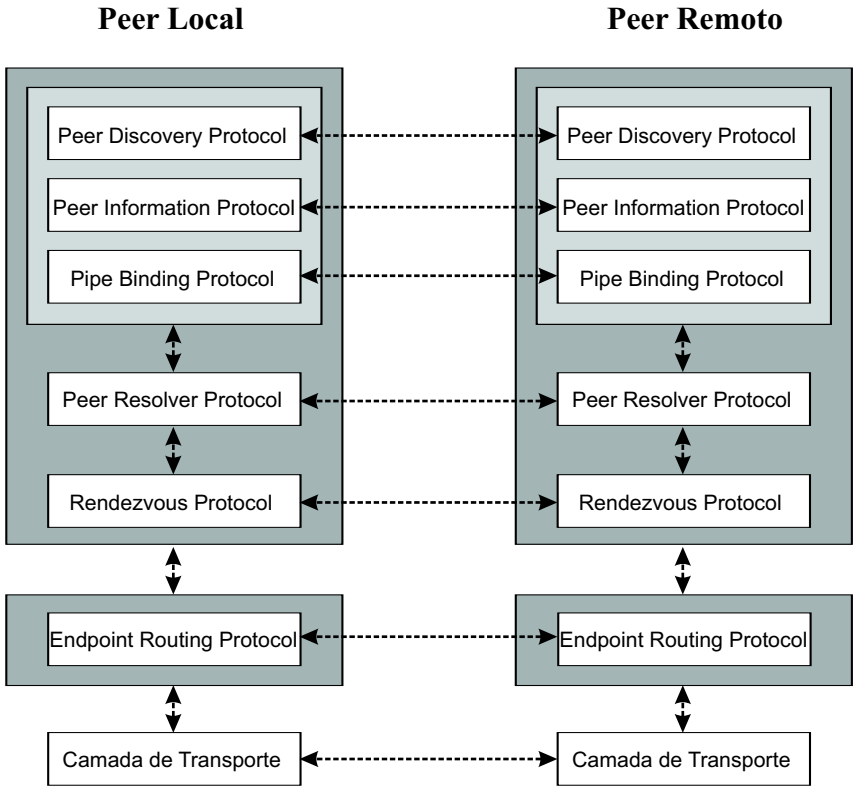


Figura 4.2. A Pilha de Protocolos JXTA

O *RendezVous Protocol* (RVP) é responsável pela propagação de mensagens a outros *peers* via *peers* especiais chamados *rendezvous*. O RVP é responsável também pelo fornecimento do serviço de *rendezvous* a outros *peers* da rede. Antes de um *peer* poder usar um *peer rendezvous* para propagar as mensagens, é necessário conectar-se ao *rendezvous* e obter um *lease*, que é uma espécie de contrato de uso do serviço. Este *lease* especifica por quanto tempo o *peer* está autorizado a usar os serviços do *rendezvous* antes de ter que renovar a conexão.

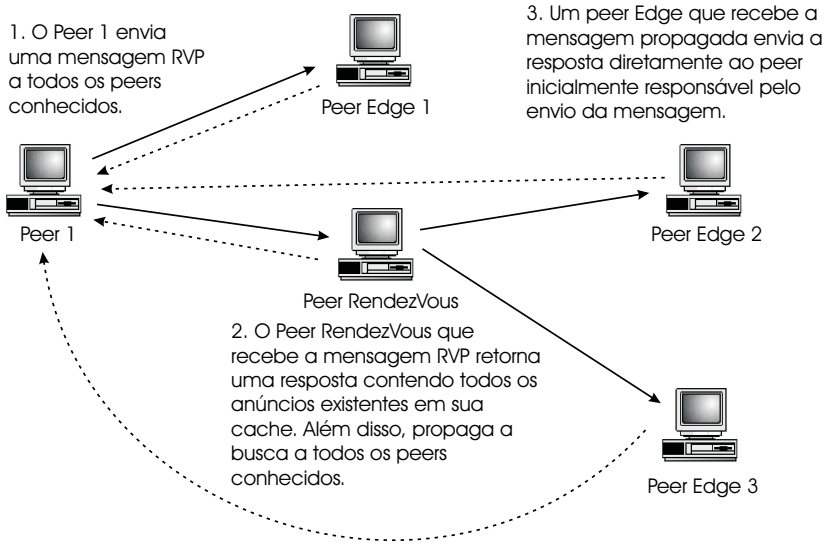


Figura 4.3. Envio de mensagens e respostas através do RVP

O *Peer Information Protocol* (PIP) permite que os *peers* possam obter informações sobre o estado de outros *peers*. Estas informações incluem o tempo de funcionamento, a quantidade de conexões, informações de tráfego e outras informações locais.

O *Peer Discovery Protocol* (PDP) define um protocolo cujo objetivo é definir como é feita a requisição de anúncios (*advertisements*) de outros *peers*, e como cada *peer* responde a uma requisição por um anúncio que foi recebida. Um *peer* descobre os recursos da rede através do envio de uma requisição a outro *peer*, usualmente um *peer Rendezvous*. Este *peer Rendezvous* é responsável por enviar a requisição adiante (atualmente através de *flooding*), e um *peer* que recebe uma requisição PDP e possui os anúncios procurados, envia a resposta diretamente ao *peer* requerente, através do ERP. O *peer* requerente recebe então um conjunto de respostas contendo anúncios que descrevem os recursos disponíveis na rede P2P. O PDP funciona de forma similar ao RVP, descrito na figura 4.3, pelo fato de utilizar os serviços do RVP.

Cada protocolo é semi-independente dos outros. Um *peer* pode escolher implementar apenas um subconjunto desses protocolos para adquirir apenas a funcionalidade básica, porém deverá confiar em seu comportamento específico a fim de eliminar a necessidade de um protocolo. Os protocolos não são totalmente independentes um do outro porque cada camada da pilha de protocolos JXTA depende da camada imediatamente abaixo. Por exemplo, embora seja possível construir uma implementação independente do PDP, não seria muito útil sem uma implementação do PRP e do ERP, a fim de efetuar o transporte de mensagens aos *peers* remotos.

Um *peer* pode também escolher implementar apenas parte de um protocolo a fim de garantir que este esteja otimizado para uma determinada tarefa. Porém, apesar de ser permitido que sejam feitas implementações parciais, a especificação JXTA recomenda que os *peers* implementem completamente todos os protocolos.

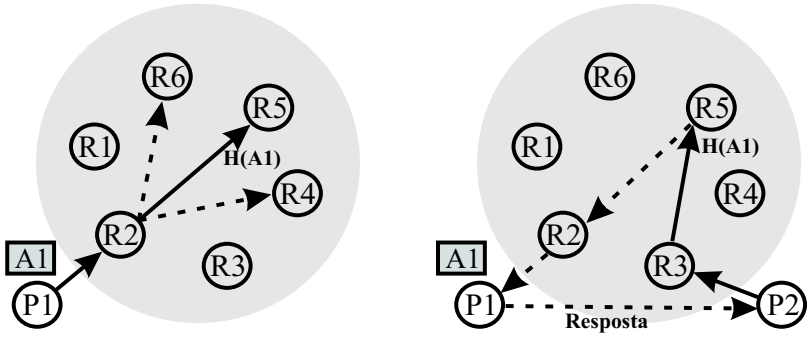
4.3 Camadas de Serviços e Aplicação

Acima de todas essas camadas existe ainda a camada de Serviços, que é uma parte desejável porém não necessária de uma solução P2P construída sobre esta plataforma. Os serviços implementam a funcionalidade que pode ser incorporada às aplicações, como a procura por recursos em um *peer*, o compartilhamento de arquivos ou autenticação de *peers*. E, finalmente, a camada de aplicação, construída sobre a camada de serviços, a fim de oferecer as aplicações P2P conhecidas. Pelo fato de que uma aplicação pode abranger apenas um único serviço ou agregar vários serviços, muitas vezes é difícil definir o que constitui a aplicação e o que constitui o serviço. Usualmente, a presença de alguma interface ao usuário indica uma aplicação, enquanto os serviços oferecem, na maior parte das vezes, uma API a ser utilizada pelo desenvolvedor.

4.4. Algoritmos de busca na plataforma JXTA

Atualmente todo o algoritmo de busca definido pela plataforma JXTA baseia-se sobre o RVP. A propagação de mensagens na rede através do RVP é feita através de *flooding* e o escopo das mensagens é controlado através do controle de *loops* e TTL (*Time-To-Live*).

Há projetos para modificação do RVP, como o algoritmo apresentado por Bernard Traversat [TRA 03]. Em seu artigo, Traversat propõe um algoritmo baseado em DHT, em que cada *rendezvous* possui uma lista interna de todos os outros *rendezvous* do grupo, ordenados por seu identificador. Esta lista é atualizada periodicamente através de troca de mensagens, de forma a manter uma certa consistência entre todas as listas existentes. A partir desta lista, é gerada então a tabela de *hash* distribuída (DHT), e cada *peer* é responsável por manter em *cache* um subconjunto das mensagens anunciadas. Quando um *peer* qualquer faz uma busca por um anúncio A, o serviço *rendezvous* faz então uma busca direta no *peer* responsável pela mensagem A, via função de hash H(A).



1. P1 Publica Anúncio A1 2. P2 Busca pelo anúncio A1

Figura 4.4. Publicação e Busca de um anúncio via DHT

A fim de manter a capacidade de recuperação da rede em caso de indisponibilidade de um dos *peers*, se este *peer* não estiver disponível, a busca é enviada

ao *peer* imediatamente acima ou abaixo na lista da DHT. Os *peers* guardam assim em sua *cache* réplicas de anúncios pertencentes a outros *peers* que são seus vizinhos na lista DHT. Esta situação é representada pela figura 4.5. em que o *peer* R5 deixa a rede, e neste momento o *peer* R6 toma o seu lugar, tornando-se o novo responsável pelas mensagens de R5.

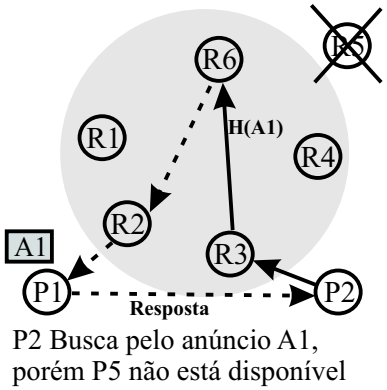


Figura 4.5. Replicação de anúncios na DHT

No caso de uma rede bastante dinâmica, em que muitos *peers* entram e saem em pouco espaço de tempo, e a lista tornar-se bastante inconsistente, o algoritmo propõe um caminhamento entre os *peers* da lista, de forma que, se um *rendezvous* não possui o anúncio requerido pela mensagem de busca, esta é enviada a seus vizinhos. Esta é a situação representada pela figura 4.6, em que dois *peers* novos entram na rede, e a DHT existente no *peer* R3 se torna inconsistente.

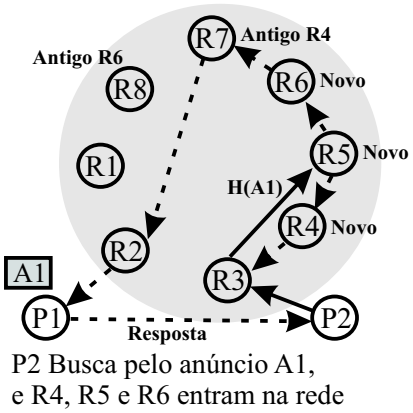


Figura 4.6. Caminhamento e busca por um anúncio em uma DHT inconsistente

Desta forma, escapa-se do *flooding*, e além disso assegura-se a disponibilidade das informações através da duplicação dos anúncios. Porém esta solução possui um problema claro: não pode ser considerada uma solução escalável. Os motivos desta afirmação são:

O custo para a manutenção da consistência da DHT são muito grandes. Se considerarmos uma lista com centenas ou milhares de *peers*, a quantidade de mensagens

– além de seu tamanho – será muito grande, tornando inviável sua aplicação para grandes sistemas.

Os custos para armazenamento da DHT para o caso de sistemas com um número grande de *peers* também é alto.

Além destes problemas que tornam esta solução não escalável, há também o fato de que todos os *peers* são tratados como se possuíssem igual quantidade de recursos de armazenamento e rede, o que também não é verdade, devido à heterogeneidade da rede. Seria interessante se pudesse ser usado um algoritmo que tirasse vantagem dessa heterogeneidade.

Apesar destes problemas, os desenvolvedores da plataforma JXTA prometem que este algoritmo estará disponível no protocolo RDV em sua próxima versão.

5. Implementação de uma aplicação sobre a plataforma JXTA

Como primeira atividade de implementação propriamente dita, foi definida uma aplicação para testar as funcionalidades da rede JXTA a ser montada. É uma aplicação leve e auto-gerenciável, sem a necessidade da intervenção do usuário, a fim de testar um grande número de *peers* ao mesmo tempo, sem a necessidade de que alguém os controle. Esta aplicação fará o papel tanto de cliente como servidor (uma típica aplicação P2P). O serviço disponibilizado é muito simples, mas suficiente para testar as funcionalidades da rede.

Os objetivos ao desenvolver esta aplicação simples é fornecer uma maneira de verificar como se comporta a rede P2P durante sua execução: conexões entre os *peers*, protocolos usados, número de mensagens enviadas, tamanho das mensagens, quantidade de tráfego, etc. As características que serão analisadas e medidas são as seguintes:

- Protocolos utilizados: quais os protocolos a nível físico e de transporte que são utilizados pela plataforma em funcionamento;
- Conexões entre os *peers*: é importante verificar como acontecem as conexões a nível de transporte entre os *peers*, para os protocolos usados: tempos de conexão, quantidade de conexões, quantidade de pacotes enviados, etc;
- Mensagens: é importante verificar também a quantidade de mensagens enviadas a nível de plataforma, o formato dessas mensagens e seu tamanho;
- Tráfego: outro ponto importante a ser analisado é a quantidade de tráfego gerado pela plataforma em funcionamento.

A aplicação desenvolvida executa um serviço em que se pode considerar que há uma grande quantidade de buscas na rede. É um serviço de disponibilização de uma lista de Títulos de Música e seus autores, com as respectivas letras de cada música. Pode ser dividida em duas partes básicas. Uma parte da aplicação é o servidor, responsável pela publicação das letras de música que o usuário local possui em seu arquivo. A outra parte da aplicação é o cliente, responsável por fazer a busca em si.

5.1. Os anúncios

A publicação é feita através do anúncio de mensagens em formato XML. A forma da mensagem esta descrita na figura 5.1.

As mensagens a serem publicadas ficam armazenadas no sistema de arquivos local do cliente, em um arquivo chamado `MUSICAS.XML`. Este arquivo é formado por um elemento raiz `<ARQUIVO>` contendo vários elementos `<MUSICA>` que possuem apenas os elementos `<TITULO>`, `<AUTOR>` e `<LETRA>` a serem publicados. Os outros elementos, `<PEERID>` e `<CONTATO>`, são preenchidos em tempo de execução pelos parâmetros fornecidos à aplicação.

```
<MUSICA>
	<TITULO>
		Título da Música
	</TITULO>
	<AUTOR>
		Autor da Música
	</AUTOR>
	<LETRA>
		Texto da letra da música
	</LETRA>
	<PEERID>
		ID do peer que fez o anúncio
	</PEERID>
	<CONTATO>
		E-mail para contato do anunciante
	</CONTATO>
</MUSICA>
```

Figura 5.1. Formato da mensagem XML a ser enviada no anúncio

5.2. O servidor

O servidor simplesmente publica os anúncios na rede P2P, aproveitando as características da plataforma para facilitar a busca. A publicação consiste no envio de mensagens aos outros *peers* da rede utilizando o mesmo algoritmo usado na busca, só que com o objetivo de fazer com que o anúncio chegue aos locais onde os outros *peers* da rede encontrarão os anúncios no momento em que fizerem a busca. Na implementação atual da plataforma, esta publicação também é feita via *fooding*, sobrecarregando a rede quando as mensagens possuem muita informação. O objetivo do *flooding* é fazer com que o anúncio chegue ao maior número possível de *peers*, de forma que possa ser facilmente encontrado. Posteriormente, na implementação a ser feita, o objetivo não deverá ser o de enviar os anúncios ao maior número possível de *peers*, mas sim de fazer com que os anúncios cheguem a um local que mais facilmente será atingido por uma busca. Isso não implica em enviar os anúncios ao maior número possível de *peers*. Implica sim em saber quais os *peers* certos a serem atingidos tanto pelo anúncio quanto pela busca.

Continuando a descrição do servidor, ele é um processo que periodicamente faz a publicação dos anúncios. Os anúncios na plataforma JXTA possuem um tempo de vida e, quando chegam à *cache* de outro *peer*, são válidos até que este tempo de vida expire. A escolha do tempo de vida relacionado aos anúncios pode ser crucial para determinar alguns aspectos do comportamento da rede. Por exemplo, se o anúncio possuir um tempo de vida muito curto, a rede poderá ficar sobrecarregada por mensagens de publicação, pois cada *peer* deverá novamente publicar seus anúncios quando seus respectivos tempos de vida expirarem. Por outro lado, se o anúncio possuir um tempo de vida muito longo, poderá haver um problema de disponibilidade dos recursos descritos por este anúncio. Por exemplo, digamos que o anúncio descreve um serviço que é disponibilizado por determinado *peer*. Se este *peer* se tornar indisponível e o anúncio continuar na rede por muito tempo ainda, esse anúncio estará descrevendo um serviço não disponível, e apenas ocupará espaço em *cache* de outros *peers* e tráfego em rede quando outro *peer* estiver buscando por esse recurso.

Por esses motivos mencionados o tempo de vida escolhido para os anúncios dessa aplicação será de 30 minutos: ao mesmo tempo em que não gera muito tráfego em rede, se o *peer* que fez o anúncio ficar mais de 30 minutos fora da rede, este anúncio

deixará de ocupar espaço na *cache* de outros *peers* e não será enviado em resposta a uma mensagem de busca em que se enquadre. O período entre cada anúncio será de 80% do tempo de vida. Não deverá ser de 100% do tempo de vida porque se o período for igual ao tempo de vida, o recurso poderá ficar por um período curto de tempo indisponível.

5.3. O cliente

O cliente é o responsável pelas mensagens de busca. Há uma interface ao usuário, em que o usuário digita uma palavra chave a ser encontrada nos campos `TITULO` ou `AUTOR`. As janelas estão descritas na figura 5.2. Achou-se também importante que houvesse além dessa interface um modo em que não seja necessária a ação do usuário. Este modo da aplicação, em que o *peer* ficaria automaticamente enviando buscas periodicamente, é importante para testar o comportamento da rede sem ser necessário que a busca esteja sendo feita por pessoas. Este modo pode então fazer o papel do usuário, e enviar buscas automáticas.Cada busca deve acontecer entre no mínimo 1 ou 2 minutos, a fim de ficar o mais próximo possível da ação real de um usuário.

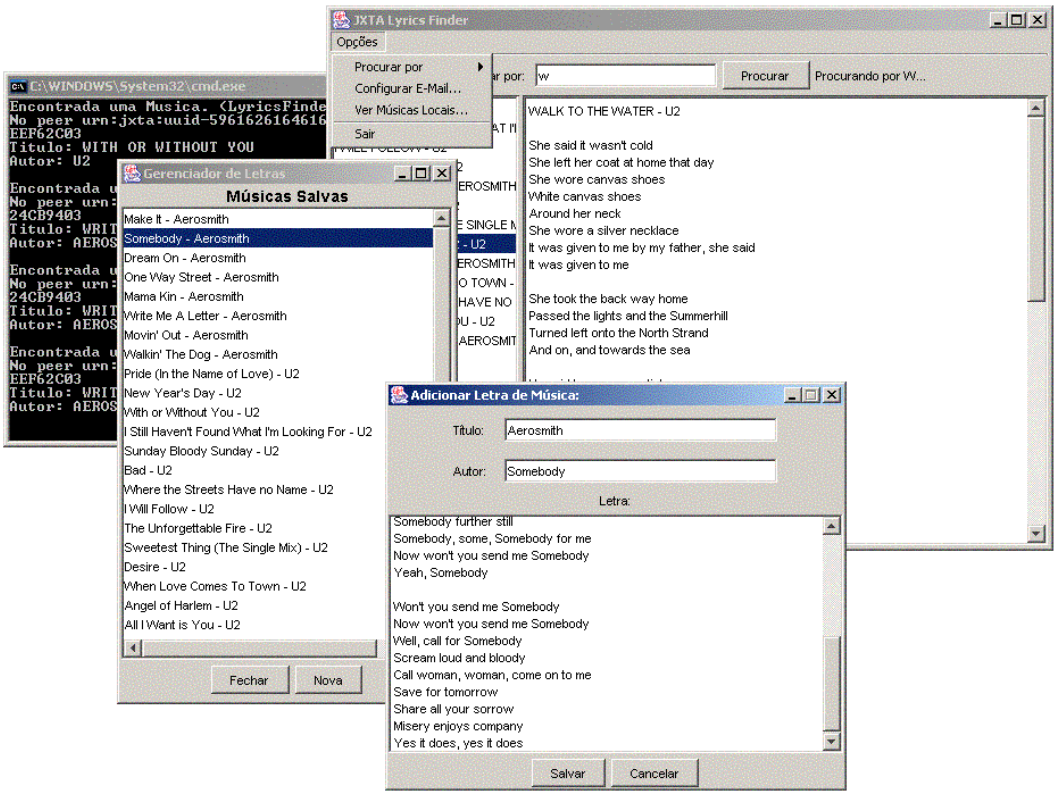


Figura 5.2. Interface Gráfica do Cliente

5.4. Análise da plataforma JXTA sem modificações

Esta análise será feita em dois passos. Primeiramente será analisado o comportamento de um *peer* executando a aplicação acima e conectado à rede JXTA pública. Após será feita a análise dos *peers* de uma rede JXTA privada, sem acesso à rede pública.

5.4.1 Análise de informações com conexão à rede pública

Primeiramente foram analisadas as informações retiradas de um *peer* conectado à rede JXTA pública. Esta rede é atualmente utilizada por desenvolvedores e pesquisadores, e portanto possui uma grande quantidade de *peers*, caracterizando-se assim em uma rede de grande escala. Nesta rede pública poderemos testar somente o funcionamento da plataforma com a atual implementação, pois a implementação feita para testar o novo algoritmo de busca tornaria os protocolos incompatíveis. A entrada na rede pública é feita pela configuração inicial do *peer*, onde é informado inicialmente um *peer rendezvous* qualquer da rede pública (*seed*) e a partir deste *peer rendezvous* é feita a busca por outros *peers* da rede pública, podendo assim inclusive conectar-se a outros *peers* diferentes. Os *peers seed* utilizados pela atual implementação são algumas máquinas do domínio jxta.org que ficam a maior parte do tempo ligadas. Após entrar na rede pública, um *peer* não precisará mais da ajuda do *peer seed*, pois guardará em sua *cache* outros *peers rendezvous* da rede aos quais poderá em outras ocasiões conectar-se inicialmente.

Após conectar-se à rede, um *peer* popula a sua *Rendezvous Peer View* (RPV) de modo a possuir em *cache* um número suficiente de *peers rendezvous* para que possa executar seu algoritmo de busca e publicação via *flooding*.

Durante o tempo de conexão da aplicação anteriormente descrita, a análise das conexões existentes a nível de protocolo de transporte (TCP) mostrou as conexões da tabela 5.1. Por estas conexões mostra-se que o protocolo de transporte utilizado foi o TCP, dado que a aplicação foi configurada inicialmente para usar somente o protocolo TCP para transporte (pode-se também utilizar HTTP se houverem configurações de *firewall* e *proxy* que não permitam conexão direta via TCP). As conexões do *peer* com outros *peers rendezvous* caracterizam-se por duplas de conexões TCP, e as conexões de *peers edge* caracterizam-se por apenas uma conexão TCP vinda do *peer edge*. Analisando a tabela chegamos à conclusão de que o tamanho da RPV durante a execução da aplicação chegou a 7 *peers rendezvous*, e que houve a conexão de um *peer edge* (220.97.41.77) no *peer* da aplicação.

	Fonte		*Destino*	
Protocolo	*Endereço IP*	*Porta*	*Endereço IP*	*Porta*
TCP	Local	1638	209.25.154.236	9701
TCP	Local	1639	209.25.154.232	9701
TCP	Local	1640	209.25.154.232	9701
TCP	Local	1641	211.19.164.106	9101
TCP	Local	1648	218.155.79.153	9701
TCP	Local	1653	211.10.20.65	9101
TCP	Local	1658	129.219.72.155	9701
TCP	66.153.100.154	61.572	Local	9701
TCP	202.32.8.225	34110	Local	9701
TCP	209.25.154.232	40130	Local	9701
TCP	209.25.154.236	46531	Local	9101
TCP	211.10.20.65	40424	Local	9701
TCP	211.19.164.106	1454	Local	9101
TCP	218.155.79.153	2348	Local	9701
TCP	220.97.41.77	3414	Local	9701

Tabela 5.1. Análise das conexões feitas durante a execução da aplicação na rede pública

A tabela 5.2 mostra a análise do tráfego gerado pela aplicação, feita pelo programa LanExplorer. Percebe-se que houveram dois *peers* com maior atividade em

relação aos outros, e esta diferença de atividade foi bastante considerável. Percebe-se também que, apesar do pouco tempo que a aplicação ficou conectada à rede – aproximadamente 20 minutos, a quantidade de mensagens foi bastante grande, gerando assim um tráfego considerável.

	Total		*Entrada*		*Saída*	
Endereço IP	*Octetos*	*Pacotes*	*Octetos*	*Pacotes*	*Octetos*	*Pacotes*
Local	4619074	8944	3834229	5316	784845	3628
211.19.164.106	3552351	6492	345278	2455	3207073	4037
211.10.20.65	448744	881	199150	399	249594	482
209.25.154.232	362788	596	82725	254	280063	342
209.25.154.236	121991	316	105312	158	16679	158
66.153.100.154	24002	80	4146	47	19856	33
202.32.8.225	21402	43	1186	19	20216	24
218.155.79.153	11887	52	8221	37	3666	15
220.97.41.77	9399	32	919	14	8480	18

Tabela 5.2. Análise de tráfego de entrada e saída
de cada um dos endereços para a rede pública

5.4.2 Análise de informações sem conexão à rede pública

Será feita a análise agora das informações geradas por 12 *peers* conectados a uma rede privada, sem acesso à rede JXTA pública. Todos os *peers* foram configurados como *peers rendezvous*. Foram inicializados cinco *peers rendezvous* a ser utilizados como *seed* pelos outros *peers*. Analisaremos também os tipos de conexão de cada *peer*, a fim de verificar se há qualquer tipo de adaptação da topologia em caso de congestionamento de alguma das conexões. A configuração inicial da rede mostrando a qual *seed* cada *peer* será inicialmente conectado é descrita pelo grafo da figura 5.3. Os *peers* são divididos em 2 grupos, dependendo da sua capacidade de transferência. Os *peers* representados por círculos mais escuros possuem capacidade de transferência da ordem de 10 Mbits/s, enquanto os *peers* representados por círculos mais claros possuem capacidade de transferência da ordem de 100 kbits/s (modens por exemplo). As arestas mais finas representam uma conexão direta entre dois *peers* da ordem de 100 kbits/s, e as arestas mais largas representam conexão direta entre dois *peers* da ordem de 10 Mbits/s (Ethernet por exemplo). Cada *peer* receberá somente um outro *peer* como *seed*, e as arestas no grafo representam estas conexões iniciais no momento da entrada do *peer* na rede. Temos assim 5 *peers* com capacidade baixa de transmissão, e 7 *peers* com alta capacidade de transmissão.

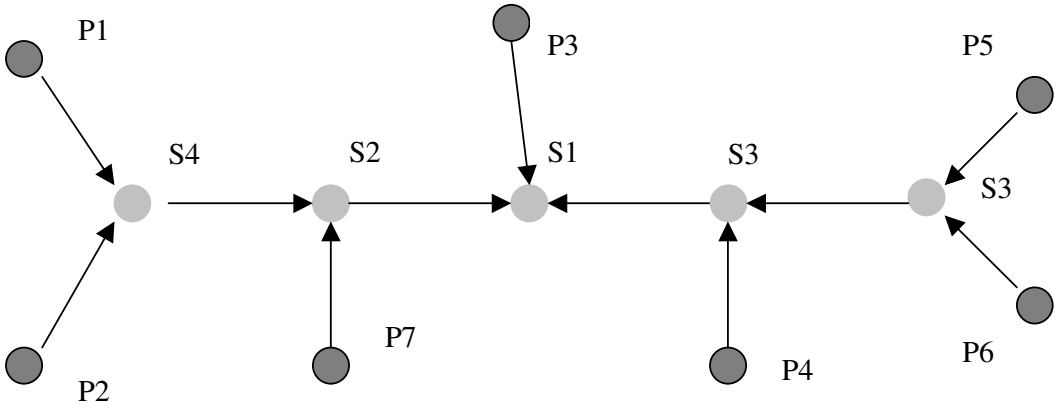


Figura 5.3. Configuração inicial da rede JXTA privada

A fim de verificar se há qualquer tipo de adaptação de topologia, a rede foi montada de forma a reproduzir o pior caso. Temos 7 *peers* com alta capacidade de transmissão, porém nenhum link com alta capacidade é inicialmente criado. Isso decorre do fato de que iniciamos a rede de tal forma que os *peers* com menos capacidade fazem parte do *backbone* da rede. O objetivo de um algoritmo de adaptação de topologia seria fazer com que os *peers* de menor capacidade fossem substituídos pelos outros de maior capacidade. No caso de a topologia da rede ficar conforme a descrita no grafo, teríamos certamente *peers* com sua conexão congestionada. Seria o caso dos *peers* S2, S3 e principalmente do S1, por estar exatamente no centro da topologia.

Após o início do funcionamento da rede, percebe-se que a topologia muda. Conforme as estatísticas colhidas, baseado nas RPVs de cada *peer* da rede, a topologia se transforma na topologia descrita pelo grafo da figura 5.4:

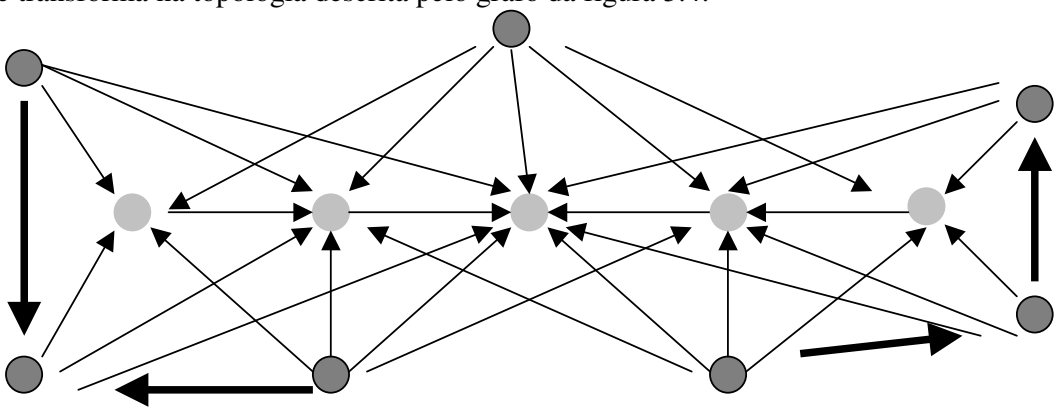


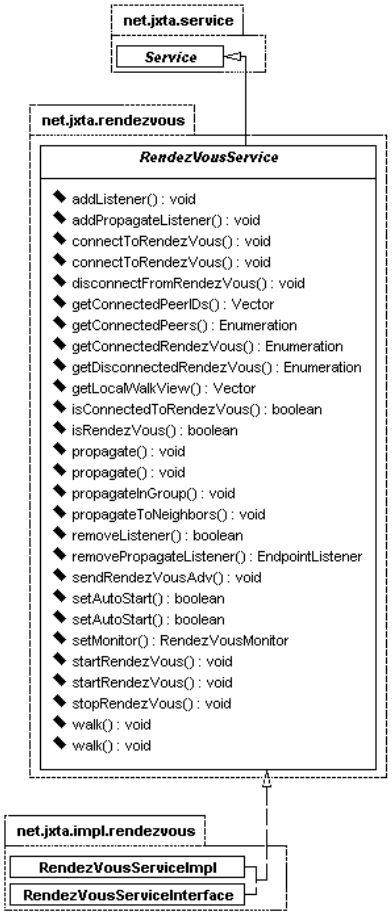
Figura 5.4. Configuração da rede JXTA privada após algum tempo

Há a descoberta dos *peers* e, conforme cada *peer* descobre os outros, adiciona-os em sua RPV. Algumas conexões diretas de mais de 100 Mbits/s se formam, mas são muito poucas se comparado com a quantidade de conexões feitas com os *peers* pertencentes ao *backbone* da rede.. Fica claro que estes *peers* tornam-se rapidamente congestionados se muitas buscas e publicações forem feitas, e a velocidade das buscas e publicações torna-se dependente da disponibilidade destas conexões. O *peer* S1 pode se tornar sobrecarregado. Uma busca feita por P1 por um anúncio existente em P6, por exemplo, deve passar por vários *peers*, quase todos com baixa capacidade. Como esta pesquisa ainda é feita via *flooding*, o congestionamento é praticamente certo.

A seguir será descrita como foi feita uma implementação do protocolo RVP utilizando um algoritmo de adaptação de topologia, e aproveitando a hierarquização feita por esta adaptação a fim de evitar o *flooding*.

6. Implementação e análise do protocolo RVP

A implementação do protocolo RVP foi feita utilizando as mesmas interfaces da plataforma, a fim de não haver nenhuma modificação nas outras camadas.



RensezVousService

A principal interface do protocolo é a interface *RendezVousService*, descrita na figura 6.1. Esta interface é acessada por outras camadas, como a cmada PRP. Os principais métodos, que se encarregam da publicação e da busca, são os métodos *propagate*, *propagateInGroup* e *propagateToNeighbors*. Porém, o método que se ocupa da propagação aos outros *peers rendezvous* em si é o método *propagateInGroup*. A interface é implementada pela classe *RendezVousServiceImpl*, que é a classe que terá seus métodos alterados a fim de ter o comportamento do algoritmo descrito.

A maior parte das classes Java existentes na implementação tradicional do protocolo RVP foi modificada. Apesar disso, muito código foi reaproveitado.

A classe utilizada pela *RensezVousService* e responsável pela manutenção das informações de cada *peer* conectado é a classe *RendezVousManager*. Esta classe possui uma classe privada chamada de *AddressElement*, que guarda as informações sobre cada *peer* nas seguintes propriedades:

- *in*: Quantidade de mensagens recebidas do *peer*;
- *out*: Quantidade de mensagens enviadas ao *peer*;
- *outMax*: Quantidade máxima de mensagens que podem ser enviadas ao *peer*.

A classe *RendezVousManager* guarda os *AddressElement* em uma árvore a fim de que os *peers* conhecidos estejam sempre ordenados. A fim de montar esta árvore, foi definida uma ordem total entre os *peers*. Esta ordem é função da capacidade de cada *peer*, e, em caso de *peers* com capacidades iguais, a ordenação é feita por seus identificadores, que são únicos. Dados dois *peers* P1 e P2, a ordem total $\leq$ entre eles é definida pelas seguintes condições:

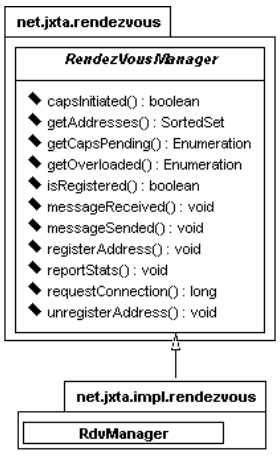


Figura 6.2. A interface *RendezVousManager*

- Se $P1.outMax \geq P2.outMax$, então $P1 \leq P2$;
- Se $P1.outMax = P2.outMax$ e $P1.id \leq P2.id$, então $P1 \leq P2$;
- Se $P1.outMax = P2.outMax$ e $P1.id = P2.id$, então $P1 = P2$.

Estas condições definem uma ordem total $\leq$ e, a partir de uma função ∂(i) decrescente, sendo *i* a colocação do *peer* em $\leq$, pode-se gerar uma distribuição de mensagens, em que os *peers* iniciais da ordem recebem mais mensagens que os *peers* finais da ordem. Esta é a grande vantagem de se ter um conjunto ordenado de todos os *peers* conhecidos: pode-se agora fazer uma distribuição de mensagens de forma que os

peers com mais capacidade recebam mais mensagens que os *peers* com menos capacidade. Esta distribuição de mensagens poderá então ser de várias formas, dependendo da função $\partial(i)$, conforme será demonstrado a seguir.

O método *propagateInGroup* é chamado no momento da criação e publicação de um recurso. a partir deste método será feito o envio das mensagens, e por isso ele será responsável por conter a função $\partial(i)$. O formato do gráfico de distribuição de mensagens entre os *peers* depende dessa função. O método *propagateInGroup* utiliza o conjunto ordenado retornado pelo método *getAddresses* da classe *RendezVousManager* para gerar esta distribuição de mensagens.

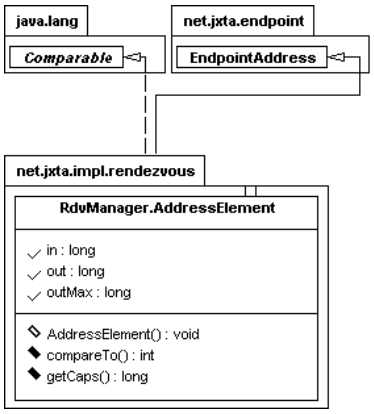


Figura 6.3. A Classe *AddressElement*

6.1. Distribuição de mensagens

A partir da função acima mencionada é possível controlar a distribuição de mensagens entre os *peers* conhecidos. Seguem agora alguns exemplos de distribuição de mensagens que poderiam ser usadas, suas vantagens e desvantagens.

Se for utilizada como função de distribuição $\partial(i) = 2^{-i}$, tem-se que a quantidade de mensagens em cada *peer* será proporcional a 2^{-i}, com i = ordem do *peer* na fila de *peers*. Utilizando essa função, os *peers* com maior capacidade recebem mais mensagens que os outros. Cada *peer* recebe metade da quantidade de mensagens do *peer* anterior. O primeiro *peer* da fila recebe metade do número total de mensagens. Como a quantidade de *peers* não é infinita, o último *peer* deve receber o restante das mensagens que sobrarem, resultando em que o último *peer* recebe a mesma quantidade de mensagens que o penúltimo *peer* da fila. O gráfico de distribuição de mensagens fica exponencial decrescente, conforme a distribuição do gráfico 6.1. Esta distribuição pode ser interessante se quisermos que os primeiros *peers* da ordem recebem muito mais

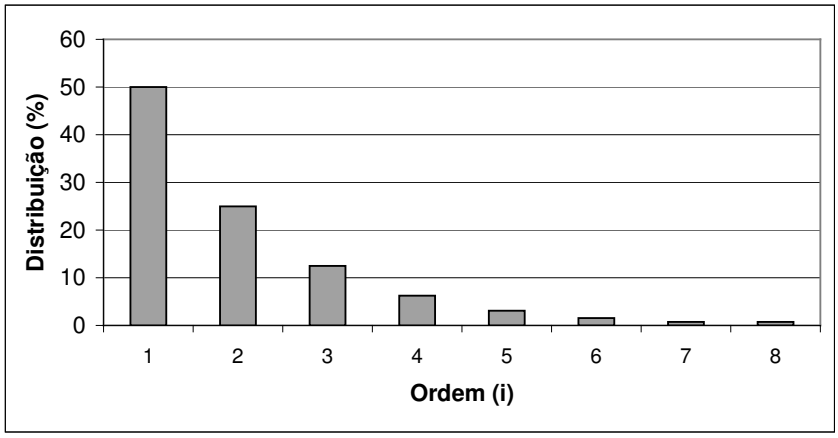


Gráfico 6.1. Distribuição exponencial de mensagens para cada *peer*

mensagens do que os últimos, e de acordo com suas capacidades, se for considerado que normalmente as capacidades dos primeiros *peers* da fila podem ser da ordem de Mbits/s, enquanto a capacidade dos últimos *peers* da fila são da ordem de Kbits/s.

Uma segunda opção de distribuição de mensagens seria uma distribuição mais equilibrada, com uma descida linear da quantidade de mensagens até o último *peer* da lista. Esta distribuição está exemplificada no gráfico 6.2, adquirido através da função $\partial(i) = \frac{n - i + 1}{P}$, onde n é a quantidade de *peers* da lista e P é o somatório $\sum_{n=1}^{p} n$. Se calcularmos o valor de P, chega-se a P = $\frac{n + n^2}{2}$, o que leva à conclusão de que $\partial(i) = \frac{2}{n}\left(1 - \frac{i}{n+1}\right)$. No caso do gráfico 6.2, n = 8.

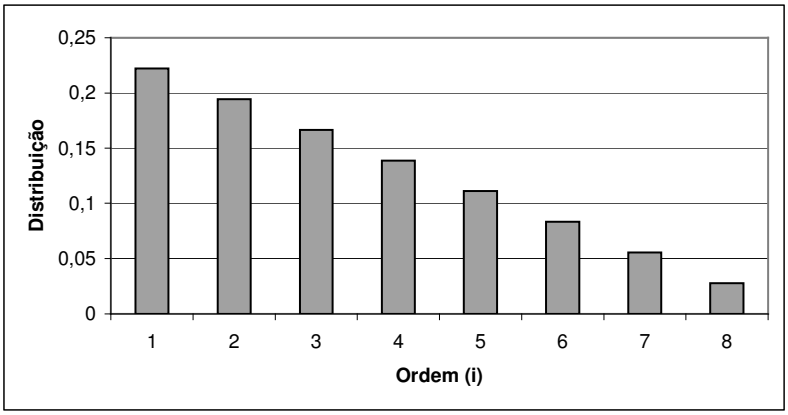


Gráfico 6.2. Distribuição linear de mensagens para cada *peer*

Uma terceira opção seria fazer a quantidade de mensagens inversamente proporcional ao quadrado da colocação do *peer* na lista. A função seria para cada pi pertencente à lista, a porcentagem de mensagens obedece à função $\partial(x) = \frac{1}{i} - \frac{1}{i+1}$, ou seja, $\partial(x) = \frac{1}{i^2 + 1}$. Esta distribuição é interessante pelo fato de que sua implementação é simples, pois pode ser implementada através da comparação de i (número da iteração) com $\frac{1}{i+1}$. O problema é que, pelo fado de a quantidade de *peers*

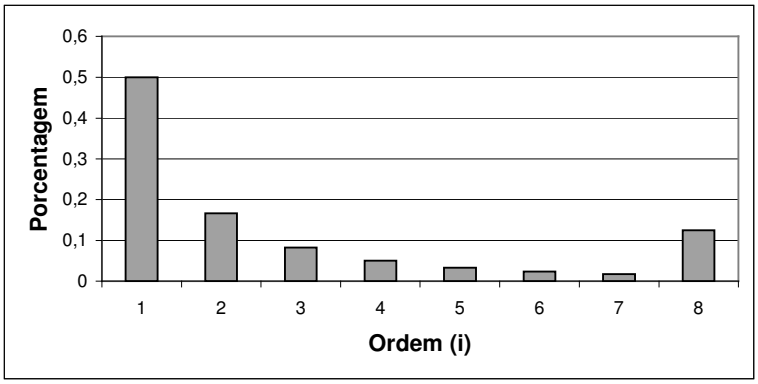


Gráfico 6.3. Distribuição de mensagens recebidas proporcional ao inverso do quadrado da ordem i

não ser infinita, o último *peer* sempre deve ficar com as mensagens restantes, resultando em um gráfico com muitas mensagens alocadas ao último *peer*.

A quarta e última opção é uma distribuição que obedeça a uma curva logística decrescente, do tipo $\partial(i) = L - \frac{L}{1 + e^{-ki+a}}$, em que *L, k* e *a* são parâmetros que podem ser modificados a fim de mudar a configuração da curva. A função descrita pelo gráfico 6.4 é deste tipo, e seria uma boa alternativa para distribuição de mensagens porque concentra grande parte dos anúncios em um número definido de *peers* de alta capacidade, o que pode vir a melhorar bastante a performance do algoritmo.

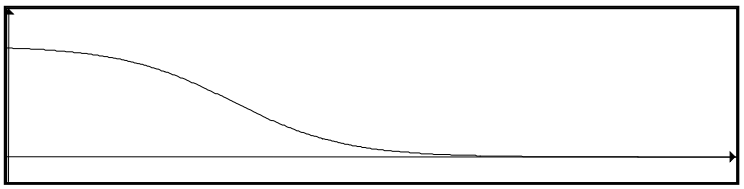


Gráfico 6.4. Distribuição logística

Bem parecida com a distribuição logística é a distribuição $\partial(i) = 1 - \frac{i^2}{i^2 + 1}$, descrita pelo gráfico 6.5, ou mesmo a distribuição $\partial(i) = e^{-i^2}$, descrita no gráfico 6.5, que poderiam ser usadas de maneira similar à distribuição logística e com as mesmas vantagens de concentração de anúncios em um número reduzido de *peers*. A única limitação para estes modelos de distribuição é que o número reduzido de *peers* que devem concentrar os anúncios deve ser suficientemente grande a fim de que o algoritmo continue escalável.

Gráfico 6.5. Distribuição $\partial(i) = 1 - \frac{i^2}{i^2 + 1}$

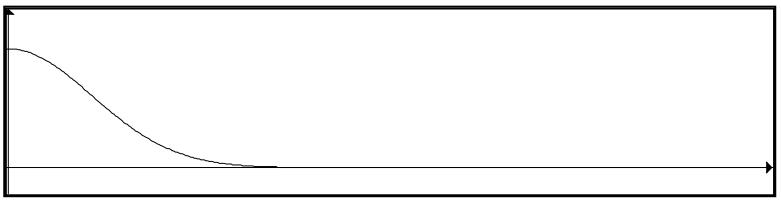


Gráfico 6.6. Distribuição $\partial(i) = e^{-i^2}$

6.2. Como é feita a distribuição das mensagens

A construção do algoritmo para distribuição das mensagens entre os *peers* foi feita da seguinte forma:

A fim de que a distribuição seja estatística, é escolhido um número randômico *r* entre 0 e 1 e encontrada uma função escolha $f(r)$ que faça a escolha do *peer* destino da mensagem. A partir do número randômico *r*, $f(r)$ retorna um valor que será o ordenamento *i* do *peer* escolhido. Essa função $f(r)$ deve ser tal que, para uma distribuição randômica de *r*, a distribuição de *i* obedeça à função $\partial(i)$ escolhida.

Por exemplo, se a função de distribuição escolhida for $\partial(i) = 2^{-i}$, dado que $\partial(i)$ é a função de distribuição acumulada, deve-se escolher uma função de distribuição não acumulada $g(i)$ cuja derivação seja 2^{-i}. Logo, $g(i) = \int \partial(i)$. Pela resolução da integração, chega-se à conclusão que $g(i) = -\frac{2^{-i}}{\ln 2}$. É preciso agora encontrar uma função que retorne *i* a partir de *r*, e não o contrário. Logo, essa função, que será chamada de $h(r)$ será igual a $g^{-1}(i)$. Através da normalização de $h(r)$, a fim de que seu domínio para *r* > 0 e r < n esteja entre 0 e 1, chegamos à função $f(r) = -\log_2(1-x)$, descrita pelo gráfico 6.7. Dado que o ordenamento *i* deve ser um número inteiro, finalmente chega-se à conclusão que função escolha é

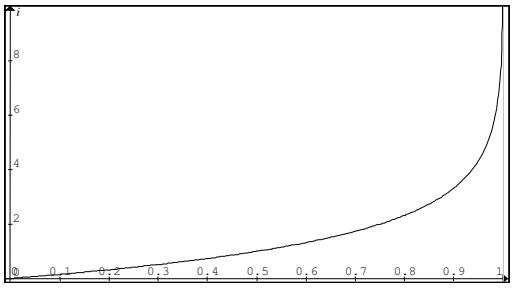


Gráfico 6.7. Função escolha de *i* para um número randômico *r* entre 0 e 1

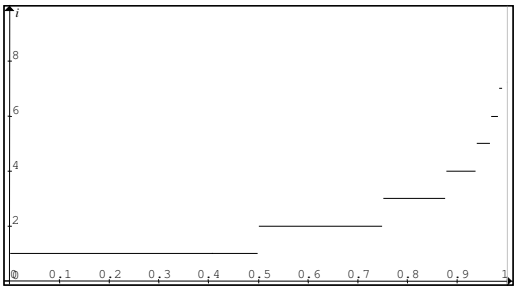


Gráfico 6.8. Função escolha de *i* retornando *i* inteiro

$i = \lceil -\log_2(1-x) \rceil$, retornando assim sempre o valor de *i* inteiro, como no gráfico 6.8.

Na escolha da função $f(r)$ para qualquer outra distribuição $\partial(i)$ deve-se seguir os mesmos passos: integração de $\partial(i)$, inversão e normalização para que o domínio fique entre 0 e 1 e retorne os valores de *i* válidos.

O resumo do algoritmo de distribuição de mensagens para busca e publicação é descrito a seguir:

```
n = número de peers;
f: função de escolha;
r = número randômico(entre 0 e 1);
peers = RendezVousManager.getAddresses(); // fila
i = f(r);
se i > n, i = n;
envia(mensagem, peers[i]);
```

6.3. Outras rotinas utilizadas pelo algoritmo

A rotina seguinte é executada cada vez que o *peer rendezvous* recebe uma mensagem de um vizinho seu:

```
se ainda tem espaço em cache, guarda a mensagem em cache;
TTL = TTL + 1
se TTL < MaxTTL
     propaga mensagem;
senão
     descarta mensagem.
```

Ao conectar-se na rede, todo *peer* executa o seguinte procedimento:

```
1. Conecta-se a um conjunto randômico de peers;
2. Para cada peer P pertencente a esse conjunto de peers:
     Ci = Capacidade de P;
     Di = Número de Peers conectados a P
     P.outMax = Ci / Di;
     P.in = 0;
     P.out = 0;
```

Esse procedimento inicializa todas as propriedades dos vizinhos do *peer*, como quantidade de mensagens recebidas, quantidade de mensagens enviadas e quantidade máxima de mensagens de cada um.

Uma questão interessante a ser analisada é como encontrar a quantidade máxima de mensagens (*outMax*) se for conhecida a taxa de transferência do *peer*. A descoberta da taxa de transferência é um outro problema que será analisado a seguir. Mas a quantidade de mensagens vai depender do tamanho de cada mensagem, que pode ser variável. Uma boa maneira de encontrar a quantidade máxima de mensagens seria conhecer o tamanho médio das mensagens, e dividir a taxa de transferência pelo tamanho médio. Teríamos então uma aproximação da quantidade máxima de mensagens. Porém essa aproximação ainda é bastante grosseira, dado que não são considerados o *overhead* de cabeçalhos IP e de outras características da rede, como a banda útil real. Outros fatores que não estão sendo considerados são:

- se o ponto de onde está sendo executada a aplicação do *peer* está sendo usado por outras aplicações que também utilizam a capacidade da rede. Neste caso a capacidade máxima que poderia ser utilizada pela aplicação *peer* é a capacidade total do ponto menos a capacidade usada pelas outras aplicações. Pior ainda será se essas outras aplicações utilizarem uma capacidade muito variável, o que torna difícil verificar o quanto está disponível ao *peer*.
- se dois ou mais *peers* estiverem dividindo o mesmo ponto de rede. É importante lembrar que um ou mais *peers* podem estar sendo executados na mesma máquina. O conceito de *peer* na plataforma JXTA não se restringe a uma máquina, com um endereço IP e um usuário, mas sim a um processo que roda nesta máquina. Pode haver mais que um desses processos rodando na mesma máquina. Neste caso, a capacidade total disponível para cada *peer* será a capacidade total do ponto dividido pela quantidade de *peers* existentes no ponto.

Outra forma de resolver este problema seria, ao invés de contar a quantidade de mensagens enviadas e recebidas por cada *peer*, verificar o tamanho de cada

mensagem em bytes e contar a quantidade de bytes enviados e recebidos por cada peer. O cálculo da capacidade máxima (*outMax*) seria então simplificado, mas os problemas anteriormente apresentados continuam.

6.4. Outras considerações sobre a implementação

Torna-se uma grande vantagem poder adaptar a topologia da rede de forma a distribuir o tráfego de uma forma planejada. Com esse planejamento, pode-se prever o comportamento da rede, e escolher a melhor distribuição para o tipo de rede escolhido. Em uma rede pequena, pode-se escolher uma distribuição linear, enquanto que em uma rede grande, pode-se escolher uma distribuição exponencial ou de curva logística.

Devido à necessidade de concentrar os anúncios no menor número possível de *peers* de forma a acelerar a busca, talvez a melhor distribuição a ser feita seja o modelo de distribuição baseado em funções logísticas. Porém, se for preferível, é possível também fazer a escolha da distribuição de forma a obedecer o mais rigorosamente possível à distribuição das capacidades na Internet. No momento não foi encontrado nenhum estudo que fizesse menção à distribuição de capacidades na Internet. Um estudo mais aproximado sobre o assunto é o feito por Faloutsos et al [FAL 99], em que é feita uma pesquisa sobre a distribuição da topologia da Internet. Seu objetivo é verificar se as distribuições dos graus de saída dos nodos da Internet obedecem alguma lei de proporcionalidade. E realmente o que se conclui é que as características da topologia da Internet obedecem às chamadas *Power-Laws*, ou leis exponenciais nas quais as distribuições das características da Internet, como grau de saída dos nodos, são proporcionais a uma função exponencial do tipo a^{-kx}.] Mihajlo A. Jovanovic, em seu trabalho [MIH 01], verifica que estas mesmas leis que se aplicam à rede Internet também se aplicam às redes P2P. Neste caso, considerando que a distribuição de capacidades também seja proporcional a uma função exponencial decrescente, e considerando que se queira fazer uma distribuição de mensagens o mais próximo possível à distribuição de capacidades, então o melhor é usar uma função exponencial decrescente também para a distribuição de mensagens.

Um dos problemas a serem resolvidos no algoritmo implementado é a descoberta automática da largura de banda disponível em cada *peer*. No protocolo que foi implementado, a capacidade de cada *peer* foi configurada manualmente, através de um arquivo de configuração no formato XML. Porém seria interessante que não fosse necessária configuração manual por parte do usuário, e que a descoberta da banda disponível para cada *peer* vizinho fosse feita automaticamente. Uma ótima maneira de implementar essa funcionalidade seria através do mecanismo proposto por Valter Roesler, Peter Finzsch, Maiko de Andrade e José Valdeni de Lima [ROE 03], o mecanismo de pares de pacotes e trens de pacotes, utilizado para descoberta de banda da rede, permitindo a inferência de sua velocidade máxima. Este mecanismo se utiliza do espaçamento em pacotes adjacentes causado pelo limite físico de banda no enlace de menor velocidade entre origem e destino, ou seja, no gargalo da rede. Através desse espaçamento e do tamanho dos pacotes, é possível inferir a largura de banda entre origem e destino.

6.5. Teste final da plataforma implementada

Durante sua execução, o protocolo implementado emite um relatório no formato XML, que resume todas as suas operações efetuadas. O relatório é gravado em um arquivo chamado `RELATORIO.XML`, encontrado no mesmo diretório do sistema de

arquivos em que estiver sendo executada a aplicação. Este arquivo possui os seguintes elementos:

```
<RELATORIO>
   <ESTATISTICAS>
      <DATAHORA></DATAHORA>
      <ENDERECO>
           <ID></ID>
           <IN></IN>
           <OUT></OUT>
           <OUTMAX></OUTMAX>
      </ENDERECO>
    </ESTATISTICAS>
</RELATORIO>
```

O elemento <RELATORIO> é formado por diversos elementos <ESTATISTICAS>, que possuem as informações de data e hora da informação coletada, e endereço de vários *peers*, com as referentes informações de identificação (<ID>), número total de mensagens que chegaram (<IN>), número de mensagens enviadas(<OUT>), e número máximo de mensagens possível ao *peer* de serem processadas(<OUTMAX>).

Após a execução completa e ininterrupta de 30 minutos da aplicação em uma rede JXTA privada, com 8 *peers*, utilizando o protocolo RVP implementado, finalmente o relatório de um dos *peers* pode ser analisado. A última instância do elemento <ESTATISTICAS>, que nos mostra a quantidade total de mensagens recebidas e enviadas a cada *peer* da rede, é apresentada abaixo:

Nome do peer	*IN*	*OUT*	*OUTMAX*
RDV1HC	285	2256	60.000
RDV2HC	141	1173	6.000
RDV1MC	60	615	1.024
RDV2MC	30	294	1.024
RDV3MC	21	144	512
RDV1LC	23	63	112
RDV2LC	20	69	112

Tabela 5.2. Estatísticas de informações de um conjunto de *peers* utilizando o novo algoritmo de busca

Aqui podemos perceber claramente que o algoritmo utilizou uma função de distribuição exponencial, dada a quantidade de mensagens de cada *peer*. Um anúncio publicado será quase sempre enviado a um número reduzido de *peers*, e a busca será feita exatamente nesses *peers*, que constituem o conjunto de peers com maior capacidade. A rede é automaticamente estruturada como uma hierarquia, em que os *peers* com maior capacidade estão em seu topo.

6.6. Trabalhos futuros

A seguir estão relacionados alguns trabalhos futuros que poderão ser desenvolvidos utilizando o trabalho até aqui feito:

6.6.1. Redirecionamento de conexões

Uma outra funcionalidade a ser implementada é o redirecionamento de conexões, descrito no algoritmo básico. Quando um *peer* da rede começa a se tornar sobrecarregado, mesmo que seja um *peer* de alta capacidade, este *peer* deve passar a redirecionar as suas conexões a outros *peers* da rede. O procedimento para verificação de sobrecarga é executado periodicamente, e é descrito a seguir:

```
Se a soma de todas as mensagens recebidas pelo peer está muito
próxima à sua capacidade total:
        Seleciona o peer P1 com maior valor em P1.in;
        Seleciona o peer P2 com maior (P2.outMax – P2.Pi.in)*,
        tal que:
               P2.outMax – P2.out >= P2.in;
               P2 ≠ P1;
               P2 não deve pertencer aos vizinhos de P1;
        Envia mensagem para redirecionar P1 a P2:
        P1 desconecta deste;
        P1 conecta a P2;
        P2.outMax = P2.outMax – P1.in;
        envia mensagem a P1 para que P1.P2.outMax = P1.in;
        envia mensagem a P2 para que P2.P1.outMax = P2.out;

*(P2.outMax – P2.Pi.in) representa a capacidade do peer P2 que
não está sendo usada. Isto significa que escolher o peer com
maior valor nesta expressão é escolher o peer com mais sobra em
sua capacidade.
```

Resumidamente, se o *peer* P está sobrecarregado, ele verifica qual é o *peer* P1 que está lhe enviando maior número de mensagens, encontra um *peer* P2 com capacidade sobrando e envia uma solicitação de redirecionamento a P1, para que a partir de agora mande suas mensagens a P2.

6.6.2. Envio de buscas paralelas

O algoritmo descrito oferece um sistema de busca por caminhamento randômico que pode retornar uma resposta após um número razoável de passos. Para diminuir esse número de passos, e fazer com que a busca retorne mais rapidamente uma resposta, é possível melhorar o algoritmo da seguinte forma: o *peer* que gera a busca, ao invés de enviar a busca a apenas um *peer*, envia a dois ou mais. O restante dos passos continua funcionando da mesma forma, e os *peers* propagam a mensagem a apenas um outro *peer*. Com o envio inicial da busca a dois ou mais *peers*, o tempo de resposta pode ser dividido por dois ou mais, pois é como se duas ou mais buscas estivessem sendo feitas paralelamente. A escalabilidade não é perdida, pois o comportamento do algoritmo no restante dos passos é o mesmo. O controle de *loops* através do identificador da mensagem ajuda a fazer com que as mensagens não repitam o mesmo caminho, aumentando ainda mais o desempenho do algoritmo.

6.6.3. Compatibilidade com o algoritmo antigo

Uma questão importante a ser resolvida é em relação à compatibilidade do funcionamento desse algoritmo com o algoritmo de *flooding* anterior. Seria possível que *peers* funcionando com este algoritmo pudessem conviver normalmente com *peers* funcionando através de *flooding* na mesma rede? A resposta parece ser sim, uma vez que os *peers* funcionando através do algoritmo de adaptação de topologia conseguem descobrir se outro peer está funcionando com o mesmo protocolo. Os dois protocolos são muito parecidos, e no protocolo do algoritmo com adaptação de topologia há apenas a adição de algumas mensagens, como requisição de capacidade máxima e redirecionamento de conexões. Se o outro *peer* não responder a essas requisições, provavelmente está funcionando através do algoritmo de *flooding*. Se uma busca ou anúncio for enviado a este *peer*, não importa de onde veio, ele executará seu algoritmo de *flooding* normalmente, e a busca ou anúncio provavelmente alcançará seu objetivo, como se nada tivesse mudado. Da mesma forma, um *peer* funcionando através do algoritmo de adaptação de topologia, recebendo uma requisição de um *peer* à la *flooding*, não mudará seu comportamento, e o anúncio ou busca alcançará seu destino. Porém, um *peer* funcionando com algoritmo de adaptação de topologia que saiba que está em uma rede heterogênea, deverá enviar suas requisições para os dois tipos de *peers*, senão correrá o risco de que suas requisições atinjam somente o grupo de *peers* que funciona com algoritmo igual ao seu.

6.6.4. Criação de um serviço dentro da plataforma JXTA

Um trabalho bastante interessante seria transformar este novo algoritmo em um serviço disponível à plataforma JXTA. Uma nova versão da plataforma com um algoritmo de busca novo baseado em DHT, como descrito no capítulo sobre a plataforma JXTA, estará disponível na próxima versão. Este algoritmo de busca estará disponível na forma de um serviço, mas que só servirá para buscas a chaves previamente conhecidas. O mesmo poderia ser feito com este algoritmo de adaptação de topologia, só que para buscas aproximadas, substituindo assim o atual algoritmo que utiliza *flooding*. Este é um trabalho que com certeza interessa à comunidade JXTA.

6.6.5. Testes em redes maiores

Os testes feitos neste trabalho se referem a redes pequenas, com baixo número de *peers*. Através da disponibilização do serviço de busca por adaptação de topologia, o serviço estaria disponível em um grande número de *peers*, e testes mais precisos em relação ao comportamento da rede poderiam ser feitos. Informações mais precisas como o quanto o tempo de resposta é aumentado, o quanto pode ser reduzido pelo envio de buscas paralelas, e gráficos mostrando o número de passos versus quantidade de mensagens encontradas seriam bastante úteis para verificar o desempenho e a possibilidade de uso real do protocolo.

Conclusões

O modelo de arquiteturas P2P é um modelo novo de arquitetura, não substituta à arquitetura cliente-servidor, mas necessário ao desenvolvimento de aplicações escaláveis. Devido ao grande crescimento da Internet, enquanto muitos sistemas cliente-servidor sofrem problemas de escalabilidade, o P2P traz grandes vantagens quanto a isso. Ainda não é um modelo maduro de arquitetura, muitas pesquisas sobre o assunto deverão ser feitas, e muitos grupos de pesquisa se interessam pela área. Com certeza, ainda existem muitas descobertas a se fazer com relação a esse assunto.

O modelo de algoritmo apresentado tem grandes vantagens por ser completamente escalável e por aproveitar as características de heterogeneidade da Internet. Aproveita também ao máximo a capacidade de cada participante da rede P2P, através de uma estrutura hierárquica auto-gerenciável e auto-adaptativa. Muitos testes ainda são necessários a fim de verificar o comportamento de uma rede de grande escala utilizando esse protocolo, e corrigir os possíveis problemas existentes.

Uma grande vantagem obtida pelo desenvolvimento do trabalho foi o aprendizado em programação Java, a utilização de muitos de seus recursos com relação à orientação de objetos que para mim ainda eram um pouco obscuros. A linguagem Java é muito poderosa em relação ao projeto de software, pois podem-se usar muitos recursos de programação a fim de aumentar a modularização da aplicação através do uso de interfaces. Essas abstrações permitem facilmente isolar a especificação da implementação, facilitando muito a construção de uma plataforma de grande escala.

Outra vantagem obtida foi o aprendizado em desenvolvimento de grandes aplicações em grupo. A boa modularização dos protocolos através da utilização dos recursos da linguagem Java e a utilização de programas para desenvolvimento concorrente de aplicações em grupo, como é o caso do CVS que foi usado, trouxe novos conceitos sobre desenvolvimento em grupo, concorrente e geograficamente distribuído.

O assunto relacionado a este trabalho demandou bastante pesquisa, e, de fato, um dos meus objetivos ao desenvolver este trabalho era não só desenvolver atividades de implementação, mas também muita pesquisa. Acredito que meus conhecimentos em relação a redes P2P cresceu muito, principalmente pelo fato de ser um assunto bastante novo, apesar de não haver muitos trabalhos relacionados, e descobri muitos conceitos aos quais não tive acesso durante o curso. A pesquisa demandada também me ajudou a crescer em minha capacidade de analisar criticamente o trabalho feito por outras pessoas, problemas relacionados, viabilidade de implementação e etc.

Enfim, graças aos conceitos tanto práticos quanto teóricos adquiridos durante todo o curso de graduação, pude aplicar muitos dos conhecimentos adquiridos, levando assim a descobertas até bastante originais, e que acredito que podem contribuir muito em relação ao assunto pesquisado.

Bibliografia

[CAR 98] CARRA, A. et al, **Jungle Monkey: Bulk File Transfer,** Distributed Systems (EECS 589) class project report, Electrical Engineering and Computer Science Department, College of Engineering, University of Michigan. Out. 1998.

[CLA 01] CLARKE, I. et al, **Freenet: A Distributed Anonymous Information Storage and Retrieval System**, Lecture Notes in Computer Science, Department of Computing, Imperial College London. Reino Unido. 2001.

[DAB 01] DABEK, F. et al, **Wide-area cooperative storage with CFS**. In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01). Out. 2001.

[DRU 01] DRUSCHEL, P.; ROWSTRON, A. **Past: Persistent and anonymous storage in a peer-to-peer networking environment**. In Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS 2001), pages 65-70, Elmau/Oberbayern, Alemanha. Maio 2001.

[FAL 99] FALOUTSOS, M. et al, **On Power-Law Relationships of the Internet Topology**, In Proceedings of SIGCOMM 99, pag. 251-262. Cambridge. Set. 1999.

[FRA 02] FRANK, A. **The Copyright Crusade II**, Viant Media and Entertainment, Jun. 2002.

[GON 01] GONG, L. **JXTA: A network programming environment**. IEEE Internet Computing, v. 5, pp. 88-95, 2001

[HEL 02] HELDER, D. A. et al, **End-host Multicast Communication Using Switch-tree Protocols,** Proceedings of the Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems (GP2PC), Maio 2002

[JXTA 03] **JXTA v2.0 Protocols Specification**, Sun Microsystems Inc. 2003. Disponível em: *http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.html*. Acesso em: julho de 2003.

[LV 02] LV, Q.; RATNASAMY, S.; SHENKER, S. **Can Heterogeneity Make Gnutella Scalable?** In: PROCEEDINGS OF THE FIRST INTERNATIONAL WORKSHOP ON PEER-TO-PEER SYSTEMS (IPTPS 2002), Mar. 2002.

[MIH 01] MIHAJLO A. JOVANOVIC B. S. **Modeling Large-Scale Peer-to-Peer Networks and a Case Study of Gnutella**, Dissertação (Mestrado em Ciência da Computação), Division of Graduate Studies and Research of the University of Cincinnati. Jun. 2001.

[RAT 02] RATNASAMY, S. **A Scalable Content Addressable Network**. Tese (Doutorado em Ciência da Computação) – Universidade da Califórnia, Berkeley – EUA, Out. 2002.

[RHE 03] RHEA, S.; EATON, P.; GEELS, D.; WEATHERSPOON, H.; ZHAO, B.; KUBIATOWICZ, J. **Pond: the OceanStore Prototype**, In Proc. of USENIX File and Storage Technologies - FAST, 2003.

[RIP 01] RIPEANU, M. **Peer-to-Peer Architecture Case Study: Gnutella Network**, Technical Report, University of Chicago. 2001.

[ROE 03] ROESLER, V.; ANDRADE, M.; FINZSCH, P.; LIMA, J. V. **Análise do mecanismo de pares de pacotes visando estimar a banda da rede via UDP**. Publicado no SBRC 2003 - 21º Simpósio Brasileiro de Redes de Computadores em Natal. Maio 2003.

[ROW 01] ROWSTRON, A.; DRUSCHEL, P. **Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems**. In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelburg, Alemanha, 2001.

[SEN 02] SEN, S.; WONG, J. **Analyzing peer-to-peer traffic across large networks**, Proceedings of ACM SIGCOMM Internet Measurement Workshop. Novembro 2002.

[STO 01] STOICA, I. et al. **Chord: A scalable peer-to-peer lookup service for internet applications**. In Proceedings of SIGCOMM 2001. Agosto 2001.

[TRA 03] TRAVERSAT, B. and al., **Project JXTA: A Loosely-Consistent DHT Rendezvous Walker**, Sun Microsystems, Inc. Mar. 2003. Disponível em: *http://www.jxta.org/project/www/docs/jxta-dht.pdf*. Acesso em: julho de 2003.

[WIL 02] WILSON, B. J., **JXTA**, New Riders Publishing, 1ª edição, Jun. 2002, 350p.

[ZHA 01] ZHAO, B. Y. et al, **Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing**, Technical Report, Computer Science Division, U. C. Berkeley. Abril 2001.