Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia
Curso de Ciência da Computação

Márcio José Martins Oliveira

Definição e implementação de uma rede P2P orientada a serviços

São Luís
2013

Márcio José Martins Oliveira

Definição e implementação de uma rede P2P orientada a serviços

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisítos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Mário Antonio Meireles Teixeira

Doutor em Ciência da Computação

São Luís

2013

Oliveira, Márcio José Martins.

Definição e implementação de uma rede P2P orientada a serviços/ Márcio José Martins Oliveira. – São Luís, 2013.

53f.

Impresso por computador (fotocópia).

Orientador: Mário Antonio Meireles Teixeira.

Monografia (Graduação) – Universidade Federal do Maranhão, Curso de Ciência da Computação, 2013.

1. Redes – Peer-to-Peer. 2. Orientação a serviços. 3. Serviços web. I. Título.

CDU 004.72

Márcio José Martins Oliveira

Definição e implementação de uma rede P2P orientada a serviços

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisítos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 07 de março de 2013

BANCA EXAMINADORA

Prof. Dr. Mário Antonio Meireles Teixeira (Orientador)

Doutor em Ciência da Computação

Universidade Federal do Maranhão

Prof. Msc. Maria Auxiliadora Freire

Mestre em Ciência de Engenharia

Universidade Federal do Maranhão

Prof. Msc. Carlos Eduardo Portela Serra de Castro

Mestre em informática

Universidade Federal do Maranhão

Aos meus pais.

Resumo

Este trabalho mostra uma solução para o problema de interoperabilidade entre as redes peer-to-peer (P2P) utilizando o paradigma orientado a serviços. Faz-se uma introdução aos conceitos básicos das redes P2P, ao protocolo Gnutella v0.4 e ao paradigma de orientação a serviços, define-se um protocolo de rede P2P utilizando a arquitetura orientada a serviços (P2PWS) e uma aplicação cliente implementando este protocolo.

Palavras-chaves: Orientação a serviços, Serviços Web, Redes P2P, Compartilhamento de arquivos.

Abstract

This work presents a solution to the problem of interoperability between peer-to-peer (P2P) networks using service-oriented paradigm. It is presented an introduction to the basic concepts of P2P networks, to the Gnutella v0.4 protocol and the service-oriented paradigm. It is defined a p2p network protocol using the service-oriented architecture (P2PWS) and a client application implementing this protocol.

Keywords: Service orientation, Web Services, P2P Networks, File sharing.

Agradecimentos

À minha namorada Cleres, ao meu amigo Vinicius e ao meu professor orientador e amigo, Mário Bros. Para estas pessoas não há palavras que expressem a minha gratidão pela conclusão deste trabalho.

"A mente que se abre a uma nova idéia
jamais voltará ao seu tamanho original."
A. Einstein

Sumário

Lista de Figuras

1 Introdução

A Internet é a maior rede de comunicações em escala mundial e sempre teve como proposta principal promover a liberdade, que deve se traduzir no acesso irrestrito a todos os recursos da rede, de qualquer lugar e a qualquer hora. Apesar disso, a Web ainda está presa ao modelo cliente/servidor, no qual servidores centralizados executam tarefas para clientes distribuídos. Ou seja, a maior parte das máquinas participa da Web apenas como coadjuvantes, acessando recursos providos por uma minoria (ROCHA et al., 2004)

A tecnologia P2P (peer-to-peer) surge como uma alternativa ao paradigma existente, à medida que não depende de uma organização central ou hierárquica e possibilita uma igualdade de funções para seus membros. O Peer-to-Peer (ou par-a-par, em português), é uma arquitetura de sistemas distribuídos caracterizada pela descentralização das funções na rede, onde cada nó realiza tanto funções de servidor quanto de cliente. Isto significa que – através dessa tecnologia – qualquer dispositivo pode acessar diretamente os recursos de outro, sem nenhum controle centralizado. Um nó (ou peer) representa cada ponto de inter-conexão com a rede, independente da função do equipamento representado por ele (podem ser computadores, PDAs, celulares etc).

Os pontos fortes das redes P2P em relação ao modelo cliente/servidor tradicional são a sua escalabilidade, isto é, lidam bem (eficientemente) tanto com grupos pequenos quanto com grupos grandes de participantes e por não possuírem ponto central de falhas, resistindo melhor a ataques maliciosos como os de negação de serviço. As redes P2P se tornaram muito populares através de aplicações como o Napster, eMule, Kazaa, LimeWire, etc, que possibilitavam aos seus usuários compartilhar recursos com outros participantes, muito embora essas mesmas aplicações tenham contribuído para associar as redes P2P à pirataria de software e de conteúdo multimídia, devido ao uso que se fez (e ainda se faz) delas.

Um problema comum dessas aplicações é a falta de interoperabilidade entre as mesmas, principalmente devido às incompatibilidades dos metadados e das interfaces das operações utilizadas na comunicação entre os nós. Os usuários, às vezes, têm em suas máquinas vários clientes de rede P2P, o que muitas vezes os força a ter várias cópias do

mesmo arquivo armazenado em seu disco, devido a configurações exigidas para cada rede. (RAMOS, 2009)

A orientação a serviços é vista como o próximo grande passo da computação distribuída (PAPAZOGLOU, 2003). Ela encapsula as lógicas de execução em serviços definidos por contratos. A aplicação de Serviços Web como serviços bem definidos no paradigma orientado a serviços potencializa o nível de abstração e reuso dos serviços, e permite uma separação dos serviços de qualquer tecnologia proprietária (ERL, 2007).

Neste contexto, justifica-se um trabalho de pesquisa que integre estas duas tecnologias: uma rede P2P orientada a serviços (P2PWS) para, além do compartilhamento de recursos e serviços, contornar problemas de interoperabilidade e facilitar a extensibilidade da rede, tornando-a fácil de ser utilizada por diversas aplicações P2P.

O objetivo geral deste trabalho é definir uma rede P2P utilizando serviços web e criar uma aplicação cliente implementando esta definição. Para isso será necessário definir uma arquitetura de rede P2P e verificar a viabilidade da utilização de serviços web na construção da mesma, definindo-se a interface de serviços web oferecida pela rede.

Inicialmente, no Capítulo 2, serão abordados todos os conceitos básicos para o desenvolvimento e entendimento deste projeto: Redes P2P, protocolo Gnutella e paradigma orientado a serviços.

O Capítulo 3 descreve a rede P2P desenvolvida nesse trabalho de monografia, passando pela definição de sua arquitetura, interface do serviço web desenvolvido, interações entre os nós e interface gráfica da aplicaçao cliente.

Por fim, no Capítulo 4, chega-se a conclusão a respeito deste trabalho e apresentam-se algumas sugestões para melhoria de suas funcionalidades.

2 Fundamentação Teórica

2.1 Redes P2P

A rede mundial de computadores sempre teve como proposta principal promover a liberdade, que deve se traduzir no acesso irrestrito a todos os recursos da rede, de qualquer lugar e a qualquer hora. Apesar disso, a Web ainda está presa ao modelo cliente/servidor, no qual servidores centralizados executam tarefas para clientes distribuídos. Ou seja, a maior parte das máquinas participa da Web apenas como coadjuvantes, acessando recursos providos por uma minoria (ROCHA et al., 2004).

A tecnologia P2P (peer-to-peer) surge para mudar o paradigma existente, à medida que não depende de uma organização central ou hierárquica, além de dispor aos seus integrantes as mesmas capacidades e responsabilidades (PARAMESWARAN; SUSARLA; WHINSTON, 2001). Através dessa tecnologia qualquer dispositivo pode acessar diretamente os recursos de outro, sem nenhum controle centralizado.

O Peer-to-Peer (do inglês: par-a-par), é uma arquitetura de sistemas distribuídos caracterizada pela descentralização das funções na rede, onde cada nó realiza tanto funções de servidor quanto de cliente. Um nó representa cada ponto de inter-conexão com a rede, independente da função do equipamento representado por ele (podem ser computadores, PDAs, celulares etc).

As redes P2P são muito populares atualmente, principalmente quando se deseja buscar ou compartilhar uma grande quantidade de informações e recursos entre os seus participantes. Porém uma das suas grandes dificuldades é a falta de interoperabilidade entre as diversas arquiteturas de redes existentes, principalmente devido às incompatibilidades dos metadados e das interfaces das operações utilizadas na comunicação entre os nós. (RAMOS, 2009).

Outro ponto crítico das redes P2P em geral é a presença dos nós mal-intencionados. Estes nós são especializados em atrapalhar as operações da rede, gerando inundação de requisições, adulteração de pacotes, respostas erradas, envio de arquivos corrompidos ou

mesmo vírus, etc. Existem algumas soluções para contornar esse problema, tal como a utilização de protocolos de reputação. Este tema é bem abordado no trabalho (RAMOS, 2009).

2.1.1 Arquiteturas das redes P2P

Basicamente, as redes P2P possuem três tipos de arquiteturas: as centralizadas, as redes com uso de super-nós e as redes P2P descentralizadas (ditas "puras").

2.1.1.1 Redes P2P centralizadas

Nas redes P2P centralizadas (Figura 2.1) predomina a existência de um servidor central, que gerencia os recursos e funcionalidades da rede, estando todos os nós da rede conectados a este servidor.

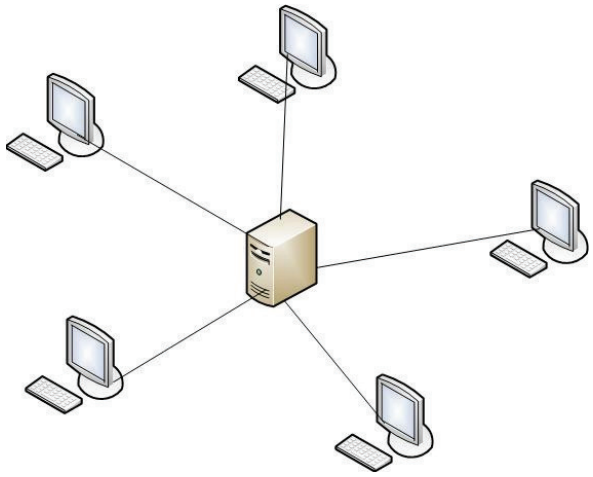

Figura 2.1: Exemplo de rede P2P centralizada

Embora esta abordagem seja simples de implementar, ela não é escalável e cria um ponto de falha e um potencial gargalo na rede. A escalabilidade desta abordagem é dependente do poder computacional do servidor central e, se o mesmo parar de funcionar, a rede também se tornará inativa.

2.1.1.2 Redes P2P descentralizadas

Nas redes P2P descentralizadas, a total liberdade é a principal característica. Nesse tipo de arquitetura não existe um servidor central ou qualquer tipo de hierarquia dos nós (é

dita P2P "pura" por isso). Cada nó pode estar ligado com qualquer outro nó da rede e as mesmas funcionalidades estão presentes em todos os nós, como mostra a figura 2.2.

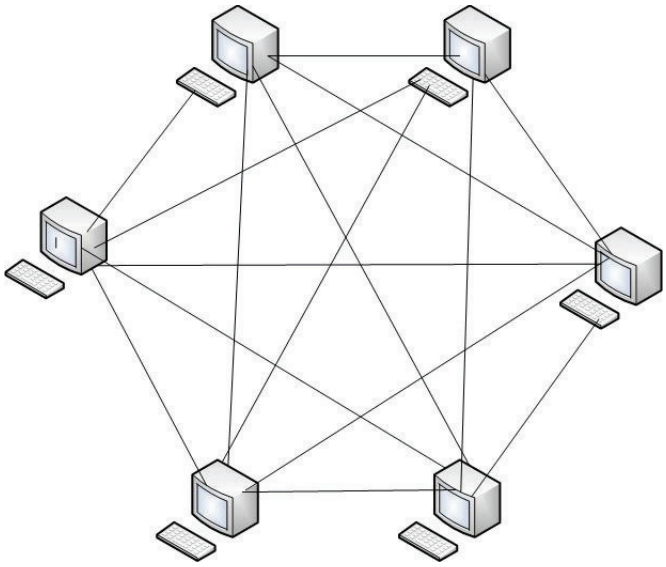

Figura 2.2: Exemplo de uma rede P2P descentralizada ou "pura"

Nota-se que, mesmo se um nó da rede deixar de funcionar corretamente, a rede não é comprometida. Portanto, as redes descentralizadas eliminam o ponto de falha encontrado nas redes P2P centralizadas, apresentando-se mais resistentes a ataques externos.

2.1.1.3 Redes P2P com super-nós

Já nas redes P2P com super-nós, a principal característica é a existência de nós na rede com atribuições de servidor. Estes nós são chamados de super-nós (Figura 2.3). Eles gerenciam sub-redes de nós e se comunicam com os demais super-nós da rede. Sendo assim, este tipo de arquitetura funciona como um híbrido entre a arquitetura centralizada e a descentralizada.

A utilização de super-nós de certa forma resolve o problema da escalabilidade das redes centralizadas. Por outro lado, este tipo de arquitetura tem a implementação mais complexa e diversos fatores estão relacionados à "eleição" de um super-nó, como sua localização geográfica, poder computacional, etc.

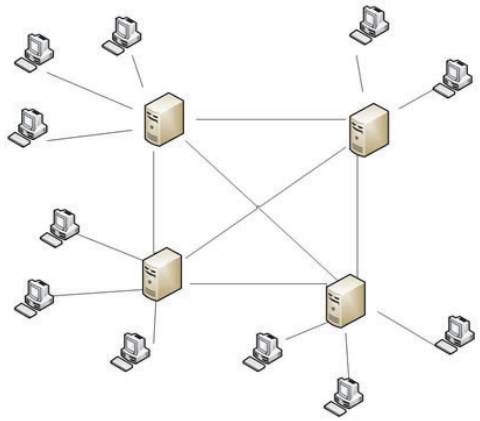

Figura 2.3: Exemplo de rede P2P com super-nós

2.1.2 Tipos de busca nas redes P2P

A busca de recursos nas redes P2P pode ocorrer de três maneiras: Busca centralizada, busca descentralizada desestruturada e descentralizada estruturada.

Na busca centralizada, é enviado um pedido de busca a um nó central – responsável pelas buscas – e este pesquisa se o recurso existe e onde o recurso se encontra. Note que as redes P2P puras não contém esse modo de busca. É importante também não misturar o conceito de arquitetura centralizada e busca centralizada. Uma arquitetura centralizada pode muito bem realizar busca desestruturada, como é o caso do Napster.

Na busca descentralizada desestruturada, quando um nó busca por um recurso ele envia uma solicitação aos seus vizinhos diretos. Estes, por sua vez, repassam a solicitação aos seus próprios vizinhos. A busca se dá por encaminhamentos sucessivos. O nó que origina a mensagem de busca também gera um número máximo de saltos (repasses), para que a mensagem não fique sendo repassada indefinidamente pela rede. Os nós que possuem o recurso procurado respondem ao nó solicitante. Este tipo de busca é utilizado no Gnutella v0.4.

Na busca descentralizada estruturada, existe uma tabela de dispersão distribuída (DHT). No modelo, tanto os dados quanto os nós recebem identificadores (únicos) de um espaço de identificadores possível (LOPES, 2010). Essa tabela hash é dividida pela rede de forma que, cada nó, dependendo de sua colocação no sistema e de seu identificador, fica responsável por uma parte da mesma. A grande dificuldade é desenvolver um algoritmo determinístico para mapear únicamente a chave de um item para o identificador do nó responsável.

2.1.3 Gnutella v0.4

É um sistema de compartilhamento de arquivos, baseado na tecnologia peer-to-peer, de topologia ad-hoc. Sua arquitetura é descentralizada (pura) e sua busca é descentralizada desestruturada. Deste modo, todos os nós do protocolo Gnutella são exatamente iguais, todos possuem as mesmas funcionalidades.

Os nós desta rede também são chamados de *servents*, pois desempenham papel tanto de cliente quanto de servidor. Eles possuem interfaces cliente através das quais os usuários podem realizar buscas e ver resultados de consultas que realizaram, enquanto ao mesmo tempo também podem responder às solicitações de outros servents, verificar se possuem o recurso procurado e disponibilizar tal recurso para download.

A busca é feita por inundação e possui um campo chamado TTL (Time To Live), que limita o número de saltos que a mensagem pode dar. Os nós que possuem o arquivo solicitado respondem ao nó que encaminhou a solicitação, sendo que a resposta à busca também segue o mesmo caminho da solicitação. O nó requisitante então escolhe um dos nós que retornaram a resposta e faz o download diretamente do mesmo.

2.1.3.1 Definição do protocolo

O protocolo Gnutella define uma maneira para os servents se comunicarem através da internet. Basicamente, existem cinco mensagens no protocolo capazes de prover as funcionalidades requeridas pela rede, a tabela 2.1 mostra estas operações.

Um Gnutella servent conecta-se à rede através do estabelecimento de uma conexão com um outro servent atualmente na rede. A aquisição do endereço de outro servent não faz parte da definição do protocolo.

Uma vez que o endereço de outro servent tenha sido obtido, uma conexão TCP/IP entre os servents é criada e a seguinte mensagem de conexão é enviada: **GNUTELLA CONNECT/<string contendo a versão do protocolo> \n \n** onde a string contendo a versão do protocolo está definida para ser ASCII "0.4". O nó que deseja aceitar a conexão deverá responder com um **GNUTELLA OK \n \n**

Qualquer outra resposta indica que o servent não deseja aceitar a conexão. Depois que o servent estiver conectado à rede gnutella, ele se comunica com os demais servents

Tabela 2.1: Descritores do protocolo Gnutella v0.4

Mensagem	**Descrição**
Ping	Utilizado para descobrir outros nós na rede. Um servent recebendo uma mensagem ping é esperado responder com um ou mais descritores Pong.
Pong	A resposta ao Ping. Inclui o endereço de conexão do servent e informação referente aos dados que o mesmo está compartilhando na rede.
Query	O mecanismo primário para busca distribuida na rede. Um servent recebendo um Query irá responder com um QueryHit se acontecer um casamento entre a busca e os seus dados compartilhados.
QueryHit	A resposta ao Query. Este descritor provê ao solicitante informações sobre como obter os dados em que ocorreram casamento com a busca.
Push	Mecanismo que permite servents que estão sob um firewall contribuir com arquivos para a rede (disponibilizar para download).

utilizando os descritores definidos pelo protocolo (Tabela 2.1).

Todas as mensagens da rede (Ping, Pong, Query, QueryHit e Push) são precedidas pelo cabeçalho mostrado na figura 2.4 e a explicação de cada campo pode ser vista na tabela 2.2.

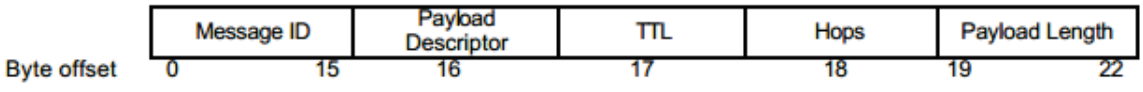

Figura 2.4: Gnutella Descriptor Header

Uma mensagem Ping no Gnutella não possui payload associado. O ping é simplesmente representado por uma mensagem de PayloadLenght zero. As demais mensagens do protocolo e seus respectivos parâmetros podem ser vistos nas tabelas 2.3, 2.4, 2.5 e 2.6.

Tabela 2.2: Gnutella Descriptor Header

Parâmetro	**Descrição**
Message ID	Uma string de 16 bytes identificando unicamente esta mensagem na rede.
Payload Descriptor	0x00 = Ping 0x01 = Pong 0x40 = Push 0x80 = Query 0x81 = QueryHit
TTL	Time To Live (Tempo de vida). O número de vezes que a mensagem será propagada até ser removida da rede. Cada servent irá decrementar o TTL antes de reencaminhar a mensagem para outro servent. Quando o TTL chegar a zero, a mensagem não será mais propagada.
Hops	A quantidade de vezes que a mensagem foi propagada pela rede. Cada vez que a mensagem é reencaminhada o hops é incrementado. O TTL e o HOPS devem satisfazer a seguinte condição: **TTL(0) = TTL(i) + Hops(i)** Onde TTL(i) e Hops(i) são os valores do TTL e do Hops no i-ésimo salto da mensagem, para i>= 0.
Payload Length	O tamanho da mensagem que seguirá imediatamente após este cabeçalho. A mensagem está localizada exatamente PayloadLength bytes a partir do fim deste cabeçalho.

Tabela 2.3: Gnutella Pong Header

Parâmetro	**Descrição**
Port	O número da porta onde o host está aceitando conexões.
IP Address	O IP do host que está respondendo.
Number of Files Shared	A quantidade de arquivos que o servent está compartilhando na rede.
Number of Kilobytes Shared	A quantidade de kilobytes que o servent está compartilhando na rede.

Tabela 2.4: Gnutella Query Header

Parâmetro	**Descrição**
Minimum Speed	A velocidade mínima (em KB/s) do servent que deve responder à essa mensagem.
Search Criteria	Uma string terminada em nulo. O tamanho máximo dessa string é cercada pelo PayloadLength do cabeçalho.

Tabela 2.5: Gnutella QueryHit Header

Parâmetro	**Descrição**
Number of Hits	A quantidade de resultados relevantes ao Query.
Port	O número da porta onde o host está aceitando conexões.
IP Address	O IP do host que está respondendo.
Speed	A velocidade (em KB/s) do host que está respondendo.
Result Set	Um conjunto de respostas correspondentes ao Query. Este conjunto contém NumberOfHits elementos, cada um deles no seguinte formato: **[File Index, File Size, File Name]** Onde File index é um número usado para identificar unicamente o arquivo, File Size é o tamanho (em bytes) deste arquivo e File Name é o nome do mesmo.
Servent Identifier	Uma string de 16 bytes identificando unicamente na rede o servent que está respondendo ao Query.

Tabela 2.6: Gnutella Push Header

Parâmetro	**Descrição**
Servent Identifier	Uma string de 16 bytes identificando unicamente o nó que está sendo solicitado a realizar o push do arquivo identificado por FileIndex.
File Index	O identificador único do arquivo a ser baixado do servent alvo.
IP Address	O IP do host que está respondendo.
Port	O número da porta onde o host está aceitando conexões.

2.1.3.2 Roteamento das mensagens

O roteamento das mensagens na rede Gnutella segue algumas regras básicas de funcionamento.

1. Mensagens Pong devem ser enviadas somente pelo caminho em que veio o Ping. Isto garante que somente os servents que rotearam o Ping verão o Pong de resposta (Figura 2.5). Um servent que receber um Pong de ID = n e não tiver originado ou visto um Ping de ID = n deve desconsiderar a mensagem.
2. O mesmo da regra 1 vale para o QueryHit em relação ao Query (Figura 2.6).
3. O mesmo vale para o Push em relação ao QueryHit (Figura 2.6).
4. Um servent irá propagar um Ping ou Query para todos os seus vizinhos diretamente conectados, exceto para o vizinho que lhe enviou o Ping ou Query atual.
5. Um servent irá decrementar o TTL e incrementar o Hops antes de propagar a mensagem para seus vizinhos diretos. Se, após decrementar, o TTL atinjir o valor zero, a mensagem não deve ser propagada.
6. Um servent que receber uma mensagem com o mesmo Payload Descriptor e mesmo Descriptor ID que já tenha recebido anteriormente deve tentar não propagar a mensagem para seus vizinhos.

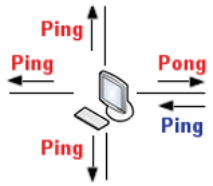

Figura 2.5: Roteamento das mensagens Ping e Pong

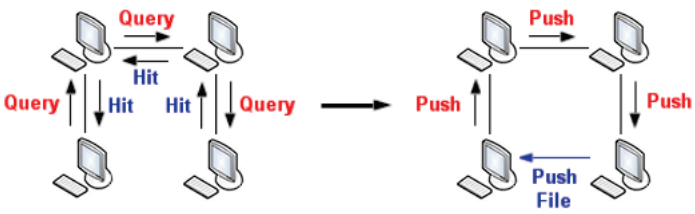

Figura 2.6: Roteamento das mensagens Query, QueryHit e push.

2.2 Serviços Web

Segundo a definição do W3C (World Wide Web Consortium), Serviço Web é um sistema de software projetado para suportar interoperabilidade máquina a máquina através de interações sobre uma rede de computadores.

Um serviço web é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Com esta tecnologia, é possível que novas aplicações possam interagir com aquelas já existentes e que sistemas desenvolvidos em plataformas diferentes sejam integrados. Em uma definição mais objetiva, serviços web são componentes que permitem às aplicações enviar e receber dados em formato XML. Cada aplicação pode ter a sua própria linguagem, que é traduzida para uma linguagem universal, o formato XML. (RAMOS, 2009)

Em termos de implementação, um serviço web é um conjunto de operações ou métodos remotos que podem ser chamados como se fossem métodos locais. A abstração da comunicação entre o cliente e o servidor (provedor do serviço) é toda feita pela plataforma, através dos padrões definidos pela mesma: *Simple Object Access Protocol* (SOAP), a *Web Services Description Language* (WSDL) e o *Universal Description, Discovery and Integration* (UDDI). Todos estes padrões serão abordados nas subseções seguintes.

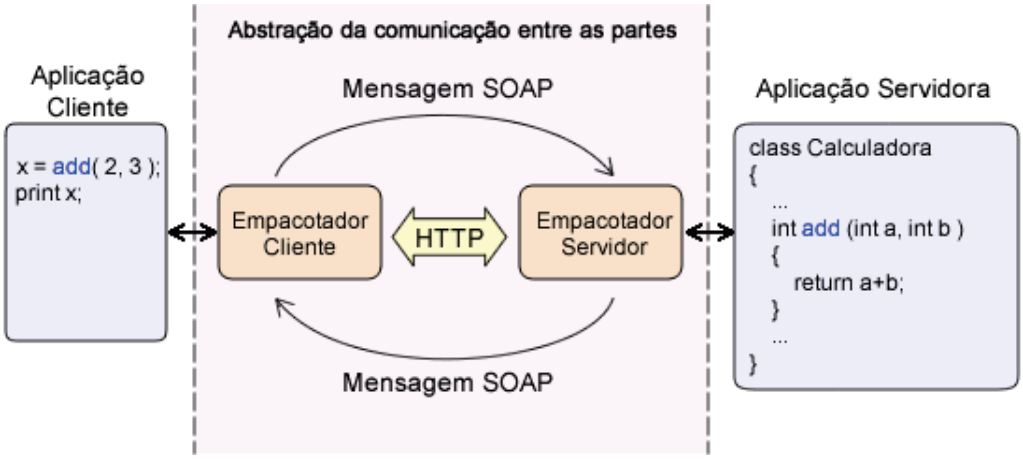

Figura 2.7: Serviço web e a abstração da comunicação.[1]

[1]Note que o método add (em azul) está definido apenas no servidor. O cliente o utiliza como um método local e toda a comunicação é realizada pela arquitetura.

2.2.1 Arquitetura SOA

A arquitetura SOA (Arquitetura Orientada a Serviços) é um paradigma que visa o desenvolvimento de softwares na forma de serviços interoperáveis. Todos os componentes de software são modelados de modo que seja possível utilizá-los em aplicações remotas, através dos protocolos da Internet. A figura 2.8 ilustra os componentes dessa arquitetura.

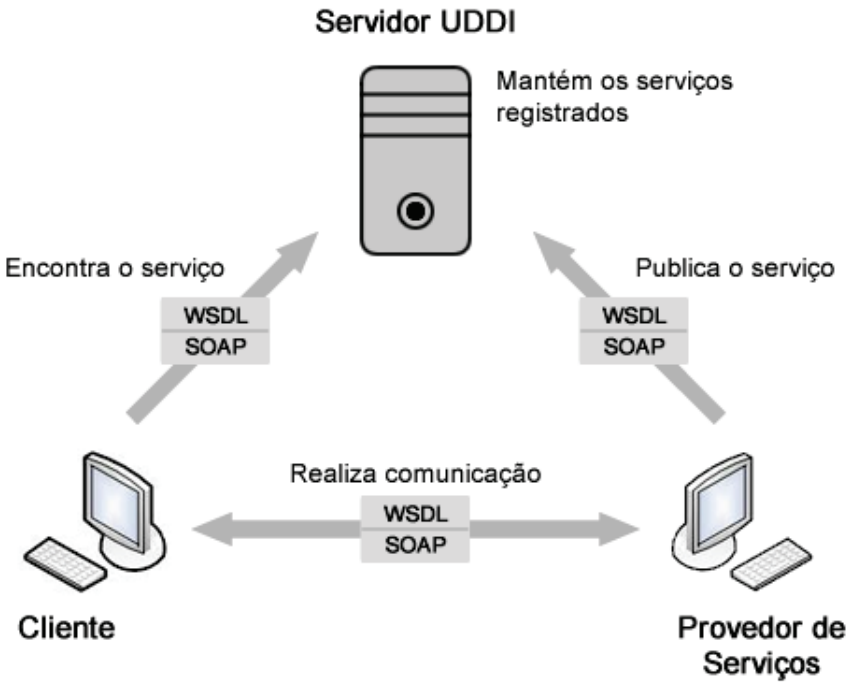

Figura 2.8: Arquitetura SOA

O provedor de serviços implementa o serviço web e pode publicá-lo no UDDI, que é um registro de serviços web. O cliente, consumidor de serviços, por sua vez pesquisa pelo serviço no repositório UDDI e, após obter a descrição do serviço, conecta-se diretamente com o provedor de serviços através do protocolo SOAP.

2.2.2 UDDI

Segundo o W3C, *Universal Description, Discovery and Integration* (UDDI) é um serviço de diretório, onde as empresas podem registrar e procurar serviços web. Basea-se em padrões da internet como XML, HTTP, utiliza a WSDL para descrever as interfaces dos serviços e comunica-se através do SOAP.

2.2.3 XML

XML é uma especificação desenvolvida pela World Wide Web Consortium (W3C) e é utilizada para representar e estruturar dados. Foi concebida para estruturar, armazenar e transportar informação, com foco naquilo que a informação representa.

Derivada da Standard Generalized Markup Language (SGML), que é um padrão internacional para definição de documentos auto-explicativos, a eXtensible Markup Language (XML) é uma linguagem de marcação para a criação de documentos com dados organizados de forma hierárquica, como mostra a figura 2.9. Além da legibilidade tanto para humanos quanto para computador, suporta o padrão Unicode e permite a criação de arquivos para definir sua própria estrutura (DTDs).

XML Document Example

```
<?xml version="1.0"?>
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

Figura 2.9: Exemplo de arquivo XML

O uso de XML nos Serviços Web permite uma clara separação entre a definição do serviço e sua execução. Esta separação nos padrões é intencional para que o Serviço Web possa trabalhar em qualquer sistema de software. A representação em XML dos tipos de dados e estruturas do serviço permite que desenvolvedores pensem nos dados que serão trocados sem considerar detalhes da implementação do serviço (NEWCOMER; LOMOW, 2004).

2.2.4 SOAP

Simple Object Access Protocol (SOAP) é um protocolo de comunicação baseado em XML para permitir que aplicações troquem informações através da internet. É independente de plataforma e de linguagem, simples, extensível e permite contornar barreiras de comunicação impostas por firewalls, uma vez que funciona sobre o HTTP, que é suportado

pela maioria dos navegadores de internet e servidores.

Uma mensagem SOAP é um XML simples, contendo os elementos mostrados na figura 2.10.

Figura 2.10: Estrutura da mensagem SOAP

O Envelope (elemento <Envelope>) funciona como container para os elementos cabeçalho e corpo. É a raiz do documento e sua função é identificar que o presente documento XML se trata de uma mensagem SOAP, além de indicar onde começa e termina esta mensagem.

O cabeçalho (elemento <Header>) da mensagem SOAP é opcional. Ele possui informações específicas da aplicação, como autenticação, autorização, roteamento, etc. Se o cabeçalho estiver definido, este deve ser o primeiro filho do elemento Envelope.

O corpo, indicado pela tag <Body>, deve obrigatoriamente estar presente em todas as mensagens SOAP. Este elemento carrega a informação destinada ao receptor final da mensagem, podendo ser uma solicitação, uma resposta ou documentos XML. Dentro do corpo pode haver o elemento <Fault>, que serve para transmissão de erros ou informações de status relativas à mensagem SOAP.

2.2.5 WSDL

WSDL (*Web Services Description Language*) é uma linguagem baseada em XML que descreve um serviço web. Ela especifica o local onde o serviço se encontra assim como os métodos que o mesmo disponibiliza.

Segundo (NETO, 2010), a WSDL possui as seguintes vantagens:

- Realiza uma clara separação lógica entre interface e implementação, o que facilita a criação e manutenção de serviços.
- Facilita o consumo de Serviços Web, ao informar os tipos de dados usados e as operações oferecidas em uma linguagem universal e independente de plataforma.
- Reduz a complexidade ao diminuir a quantidade de código (e erros potenciais) que uma aplicação cliente precisa implementar.
- Facilita mudanças nas implementações dos serviços. Alterações na implementação do serviço não são refletidas na interface.

A estrutura de um documento wsdl pode ser vista na figura 2.11.

```
<definitions>

    <types>
        Definições dos tipos de dados utilizados...
    </types>

    <message>
        Definição dos "parâmetros" das operações...
    </message>

    <portType>
        Conjunto de operações...
    </portType>

    <binding>
        Especificação do formato dos dados e do protocolo utilizado...
    </binding>

</definitions>
```

Figura 2.11: Estrutura de um documento WSDL

O elemento <definitions> é a raiz de um documento wsdl.

O elemento <portType> é o mais importante do WSDL. Ele descreve o serviço web, as operações que podem ser realizadas e as mensagens envolvidas. (W3SCHOOLS..., 2012).

O elemento <message> define os dados de uma operação. Cada mensagem é constituida de uma ou mais partes (Elemento <part>). Essas partes são análogas à parametros de funções em uma linguagem de programação tradicional. (W3SCHOOLS..., 2012).

O elemento <types> define os tipos dos dados utilizados pelo web service. Para maior neutralidade, WSDL utiliza XML Schema para definir os tipos dos dados. (W3SCHOOLS..., 2012).

O elemento <binding> associa a interface inteira ou apenas uma operação específica (elemento <portType>) a uma tecnologia de transporte. SOAP é a forma mais comum dessa associação. O elemento <port> representa um endereço físico no qual o serviço pode ser acessado.

O documento WSDL também pode conter outros elementos, como o elemento <service>, que torna possível agrupar as definições de vários serviços em um mesmo documento WSDL. (W3SCHOOLS..., 2012).

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

Figura 2.12: Fragmento de um documento WSDL

3 Proposta da Rede Peer-to-Peer: P2PWS

A proposta desse projeto é desenvolver uma aplicação P2P com a infra-estrutura baseada em serviços web para, além do compartilhamento de recursos e serviços, contornar problemas de interoperabilidade e facilitar sua extensibilidade, tornando-a fácil de ser utilizada por diversas aplicações de redes P2P.

Os serviços web estarão presentes na infra-estrutura básica da rede P2P definida e desenvolvida neste projeto, em operações como ingresso do nó na rede, busca de recursos e transferência de arquivos. Além disso, a rede será uma adaptação do protocolo Gnutella v0.4 para funcionar sobre a arquitetura SOA. Desse modo, o problema da interoperabilidade nas redes P2P é tratado através da definição de uma arquitetura orientada a serviços e da implementação de uma ferramenta que agregará estes dois conceitos: O P2PWS, uma arquitetura de rede P2P orientada a serviços.

Neste capítulo, serão discutidos os tópicos relacionados à definição e a implementação da arquitetura P2P desenvolvida.

3.1 Arquitetura

A arquitetura da rede P2PWS é descentralizada e não-estruturada (pura) por priorizar a igualdade das funcionalidades em cada nó e, principalmente, para que a aplicação seja independente de qualquer tipo de servidor. Isto garante que o P2PWS tenha um funcionamento contínuo, sem riscos de parar por quaisquer problemas em um único terminal da rede.

Portanto, cada nó da rede é um módulo independente e por si só provê todas as funcionalidades tanto para a formação e manutenção da rede quanto para o compartilhamento de recursos.

Um nó da rede P2PWS é representado por uma entidade chamada Peer (Figura 3.1) e possui basicamente quatro informações:

id Identificador único do nó na rede.

wsdl Endereço wsdl descrevendo os serviços oferecidos por este nó.

nFiles Quantidade de arquivos compartilhados.

nBytes Quantidade de bytes compartilhados

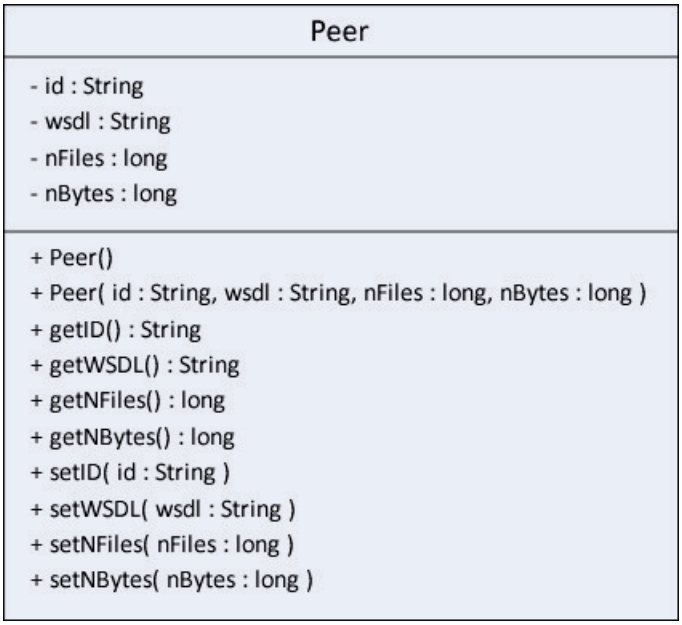

Figura 3.1: Classe Peer

Apenas os campos *id* e *wsdl* são suficientes para a formação e manutenção da rede. Porém os campos *nFiles* e *nBytes* são importantes para o desenvolvimento de protocolos de reputação, além de estarem presentes na definição do Gnutella v0.4 (ver tabela 2.3).

O endereço wsdl do nó é o seu ponto de entrada para conexão. É através dele que os nós se conhecem e se comunicam. Como visto na seção 2, o arquivo wsdl possui a assinatura de todos os métodos implementados por um serviço. A convenção adotada na rede P2PWS para o endereço WSDL dos nós segue o formato mostrado na figura 3.2

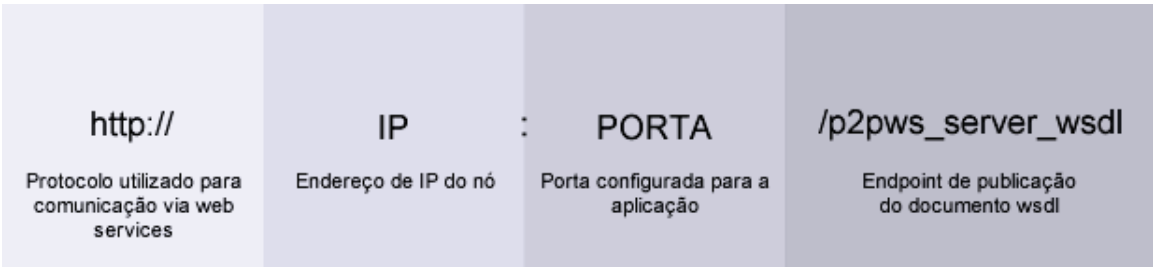

Figura 3.2: Padrão de endereçamento wsdl do P2PWS

A busca é desestruturada e realizada através de encaminhamentos sucessivos. Um determinado nó que busca por um recurso dispara uma solicitação aos seus vizinhos. Os

vizinhos deste nó, por sua vez, pesquisam se possuem arquivos relevantes à consulta em sua base de dados e, se a mensagem ainda possuir tempo de vida (TTL), também reencaminham a solicitação. Isso sucede até que a mensagem não possa mais ser encaminhada e o nó apenas consulte sua base de dados e retorne. A busca está intrinsecamente associada ao método Query e os detalhes da mesma serão abordados adiante.

Outro ponto importante é a codificação de caracteres. A busca por recursos na rede P2PWS se dá através da passagem de mensagens em forma de String. Para isso, foi definida uma codificação padrão. As strings de consulta devem ser convertidas em forma de URL (HTML..., 2012) utilizando a codificação UTF-8.

Ao codificar uma String para uma consulta, as seguintes regras se aplicam:

- Os caracteres alfanuméricos “a” a “z”, “A” a “Z” e “0” até “9” são os mesmos.
- Os caracteres especiais “.”, “-”, “*” e “_” permanecem os mesmos.
- O caractere de espaço “ ” é convertido em um sinal de mais “+”.
- Todos os outros caracteres são inseguros e são convertidos em um ou mais bytes usando o esquema de codificação UTF-8.
- Em seguida, cada byte é representado pela string de três caracteres “%xy”, onde xy é a representação hexadecimal de dois dígitos do byte.

Por exemplo, ao codificar a sequência “The string ü@foo-bar”, o resultado seria “The+string+%C3%BC%40foo-bar”, porque em UTF-8 o caractere ü é codificado como dois bytes C3(hex) e BC(hex), e o caractere @ é codificado como um byte 40(hex). (JAVA..., 2012).

A geração de ids na rede P2PWS se da através da classe IDFactory. Esta classe possui dois métodos estáticos para a geração de ids de nó (generatePeerID) e ids de mensagem (generateMessageID). O IDFactory utiliza uma classe do Java para geração de um *Identificador Universalmente Único (UUID)* e a geração desse identificador único segue a definição descrita pela RFC 4122 (A..., 2005).

3.2 Interface do serviço web

O coração do projeto está na definição dos métodos do serviço web. Basicamente, as mensagens trocadas anteriormente pelo protocolo Gnutella v0.4 agora são realizadas através de chamadas a serviços web. Em termos de implementação, não é mais necessário criar rotinas para realizar o parsing das mensagens, uma vez que toda a comunicação fica a cargo da arquitetura orientada a serviços.

A rede P2PWS define um conjunto de quatro métodos (tabela 3.1) que – em sua maioria – foram adaptados do Gnutella v0.4 para funcionar sobre a arquitetura SOA.

Tabela 3.1: Métodos do protocolo P2PWS

Método	**Descrição**
Join	Conecta o nó solicitante à rede P2PWS. Este é o método de entrada na rede. O solicitante passa suas informações de Peer e recebe uma lista de vizinhos (peers). O nó solicitado também conecta o nó recém-chegado aos demais vizinhos através do método Connect.
Connect	Conecta um nó à outro. Este é um método de atualização de informações entre as partes. O solicitante passa suas informações de Peer e recebe as informações de Peer do nó solicitado.
Query	Mecanismo primário para a busca na rede. O solicitante dispara uma consulta ao solicitado. Este, por sua vez, pesquisa sua base de dados e repassa a consulta aos seus vizinhos. O processo sucede até que a mensagem não tenha mais tempo de vida.
Download	Mecanismo primário para download na rede. O solicitante informa qual arquivo deseja baixar e qual a próxima porção de bytes a ser baixada. O solicitado retorna a próxima faixa de bytes deste arquivo e uma mensagem de status indicando a continuação, falha ou término do download.

A rede P2PWS não define um método específico para a verificação de atividade de um nó, como o método *ping* no Gnutella v0.4. Esta não-definição foi proposital pois, para testar se um nó está ativo ou não, basta realizar uma chamada a qualquer um dos métodos do protocolo. Se a chamada não obtiver resposta, significa que o nó não está disponível. Apesar disso, é recomendável o uso do método *connect* para realizar esta verificação, por

trocar informações simples e manter as informações atualizadas entre os nós.

3.2.1 Join

O método Join define uma maneira de um determinado nó se integrar à rede P2PWS. A assinatura do método pode ser vista abaixo:

public Peer[] **join**(Peer p);

O método realiza quatro operações:

1. Recebe as informações de Peer do nó solicitante (parâmetro p).
2. Adiciona o solicitante à minha lista de vizinhos.
3. Conecta o solicitante aos meus vizinhos através do método Connect.
4. Retorna ao solicitante a minha lista de vizinhos (Peers) incluindo as minhas informações de nó.

Ao fim do método, o nó solicitante receberá uma lista de vizinhos para os quais poderá realizar solicitações e todos os nós desta lista saberão da existência do mesmo na rede.

3.2.1.1 Interações do método Join

Um nó X que deseja se conectar à rede P2PWS efetua a chamada do método join para o nó A já pertencente à rede.

Para isso, passa suas informações de Peer como parâmetro do método. O nó A, ao receber a chamada ao join, adiciona o nó X em sua lista de vizinhos (passa a conhecer o nó X) e chama o método *connect* para todos os seus vizinhos, no caso, B e C, passando as informações de Peer do nó X. Isto faz com que os nós B e C conheçam o nó X. Feito isso, o nó A retorna sua lista de vizinhos(B e C) ao nó X, incluindo as suas informações de nó. No fim do método o nó X conhece os nós A, B e C e todos eles conhecem o nó X, exatamente como ilustrado na figura 3.3.

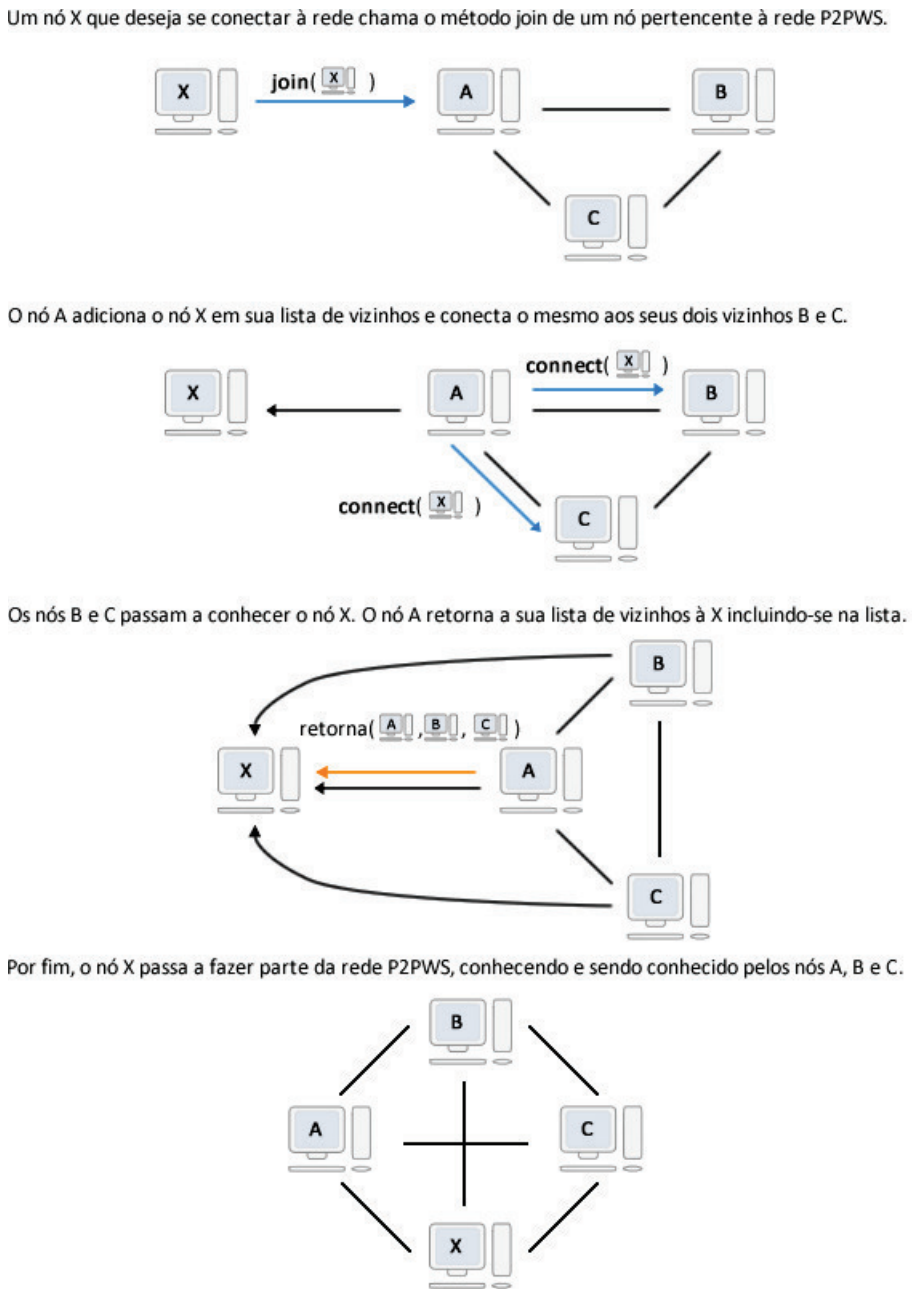

Figura 3.3: Iterações do método Join

3.2.2 Connect

O método Connect define uma forma de os nós atualizarem suas informações entre si. Assim que a aplicação é iniciada o nó em questão deve conectar-se à todos os seus vizinhos. A assinatura do método connect está descrita abaixo:

public Peer **connect**(Peer p);

O método realiza três operações:

1. Recebe as informações de Peer do nó solicitante (parâmetro p).

2. Atualiza as informações do solicitante em minha tabela de vizinhos (se o mesmo não for encontrado, adiciona-o).

3. Retorna ao solicitante as minhas informações atuais de Peer.

Ao fim do método, solicitante e solicitado possuem informações atualizadas de ambos.

3.2.2.1 Interações do método Connect

O *connect* é um método de atualização de informações. O ideal é que assim que a aplicação for iniciada o nó conecte-se a todos os seus vizinhos. Um nó X chama o método *connect* para o nó A, passando suas informações atuais de Peer. O nó A – por sua vez – atualiza as informações de X em sua tabela de vizinhos (caso X não exista na tabela de vizinhos, ele será adicionado à mesma) e retorna para X suas informações também atualizadas. Por fim, tanto X quanto A possuem informações atualizadas um do outro. A figura 3.4 ilustra o processo descrito acima.

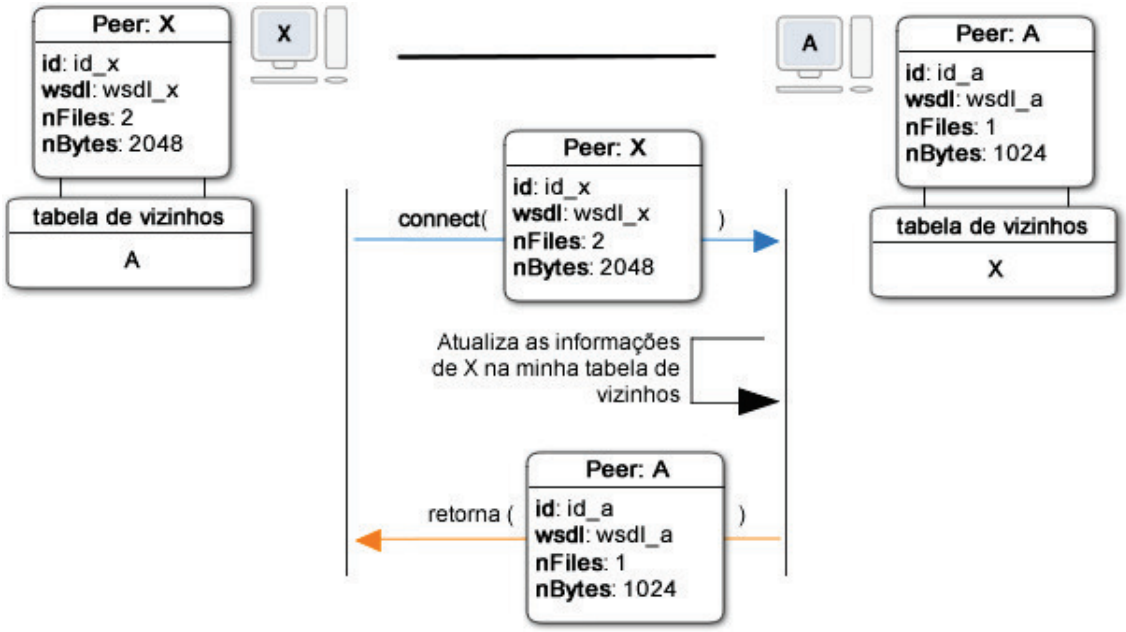

Figura 3.4: Iterações do método Connect

3.2.3 Query

O método query define um mecanismo de busca simples para a rede. A assinatura do mesmo pode ser vista abaixo e a tabela 3.2 contém as informações dos parâmetros deste método.

public Source[] **query**(String encResourceName, String messageID, String origPeerID, int TTL);

Tabela 3.2: Parâmetros do método Query

Parâmetro	**Descrição**
encResourceName	O nome do recurso a ser buscado, codificado utilizando o padrão de codificação do P2PWS descrito na seção Arquitetura deste capítulo.
messageID	O ID gerado para a mensagem. É útil para evitar o reprocessamento de mensagens duplicadas.
origPeerID	O ID do nó que originou a busca. Serve para que os nós não repassem a busca ao nó que originou a mesma.
TTL	Tempo de vida da mensagem. A mensagem é repassada aos vizinhos enquanto TTL > 0.

O retorno do método query é um array de objetos do tipo Source. Este objeto representa uma fonte de recursos relevantes a uma consulta. A figura 3.5 representa a classe Source, que possui três informações básicas:

source Um objeto da classe Peer, representando o nó da rede P2PWS que contém os arquivos listados em fileNames.

fileNames Uma lista de strings contendo o path relativo dos arquivos relevantes à consulta (codificados utilizando o padrão do P2PWS).

fileSizes Uma lista de tamanhos de arquivos (em bytes), onde o i-ésimo elemento desta lista representa o tamanho do i-ésimo arquivo em fileNames.

O método realiza uma série de procedimentos, dentre os quais está a busca pelo recurso em sua base de dados e o repasse da mensagem aos nós vizinhos. A figura 3.6 ilustra os procedimentos do método query em um pseudo-código descritivo.

A consulta pelo recurso na base de dados consiste apenas em percorrer todos os arquivos, pastas e subpastas contidos na pasta compartilhada da aplicação, procurando por nomes de arquivos que casem perfeitamente com a string de consulta ou que contenham a string consultada. Por exemplo, se a pasta compartilhada da aplicação contiver três arquivos chamados “temp.txt”, “temporal.png” e “outro.swf” e a palavra-chave da

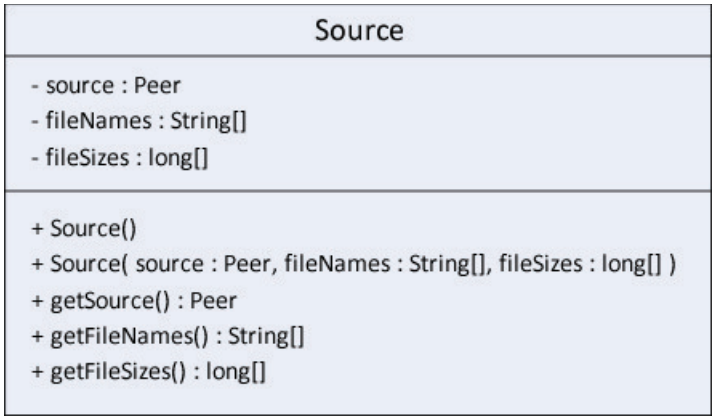

Figura 3.5: Classe Source

```
{
    ...
    Decodifica o nome do arquivo;
    Se ( jáProcessou( messageID ) ) { // esta mensagem já foi processada
        Retorna um array vazio;
    }
    Se não {
        Se ( TTL > 0 ) {
            Para cada vizinho 'i' em minha tabela de vizinhos {
                // obs: não envia a mensagem para o nó que a originou
                Se ( vizinho[i].id != origPeerID ) {
                    Chama método query para 'i', com os mesmo parametros
                    e decrementando o TTL em uma unidade;
                }
            }
        }

        Pesquisa minha base de dados pelo recurso;

        Retorna um array de objetos Source contendo os
        resultados da pesquisa em minha base de dados e
        os resultados retornados pelos vizinhos;

    }
}
```

Figura 3.6: Passos do método query em pseudo-código descritivo

consulta for “temp”, os arquivos “temp.txt” e “temporal.png” serão retornados, porque ambos contém a palavra-chave “temp”. A consulta não é case-sensitive, isto é, as strings (nome do arquivo e palavra-chave) são convertidas para minúsculas antes da comparação.

3.2.3.1 Interações do método Query

O nó X deseja buscar um arquivo “arq”. Para isso, ele dispara uma chamada ao método *query* para seu vizinho A passando como parâmetro o nome do arquivo codificado (arq), um id gerado para a mensagem (idMSG), seu id de nó (id_X) para que os nós não repassem a solicitação de volta a ele e o TTL dessa mensagem (1). Exatamente como ilustrado no quadro 1 da figura 3.7. O nó A, ao receber o *query*, chama o mesmo método em todos os seus vizinhos (repassa a mensagem) decrementando o TTL em uma unidade, como mostrado no quadro 2. Os nós B e C, ao receber a chamada ao método *query*, verificam que o TTL da mensagem é 0 (zero) e portanto não devem propagar a mensagem. Apenas pesquisam sua base de dados e retornam para o nó A uma lista de elementos do tipo Source contendo apenas o Source relativo à consulta em suas bases de dados (Source_B ou Source_C), como mostra o quadro 3 da figura 3.7. Por fim, o nó A consulta a sua base de dados buscando por recursos semelhantes à string de busca “arq” e retorna para o nó X uma lista de objetos Source, contendo a sua consulta (Source_A) e a dos vizinhos (Source_B e Source_C), como mostrado no quadro 4.

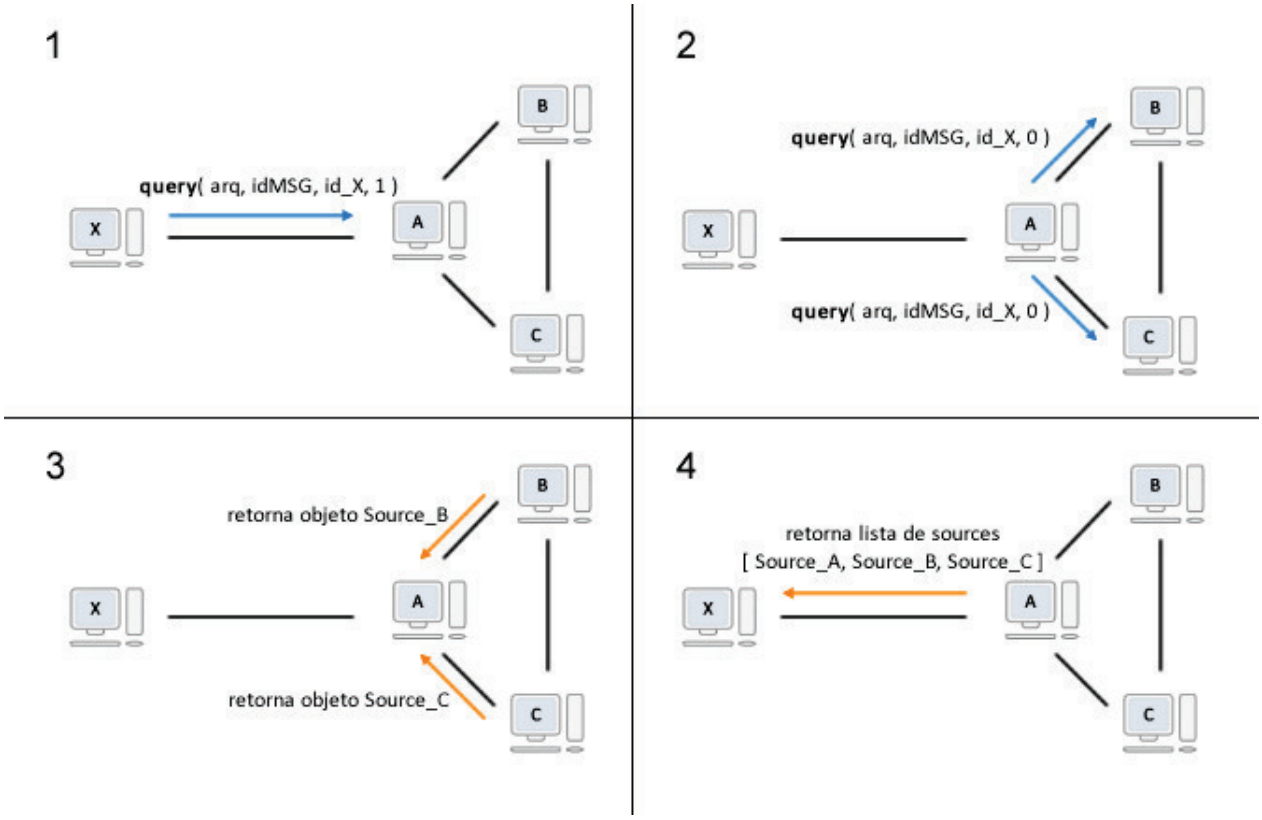

Figura 3.7: Iterações do método Query

3.2.4 Download

O método download define um mecanismo para baixar o arquivo requerido através do próprio serviço web, isto é, sem criar uma conexão para transferir todo o arquivo. O arquivo é baixado através de sucessivas chamadas ao método download, cada uma delas para baixar uma parte do recurso solicitado. A assinatura do método está descrita abaixo:

public FilePart **download**(String encFilePath, long lastFileSize);

O parâmetro *encFilePath* deve conter o caminho relativo do arquivo na máquina destino, exatamente como recebido através do método *Query*. O parâmetro *lastFileSize* contém o valor da próxima porção de bytes a ser baixada do arquivo. Ao iniciar um download, por exemplo, *lastFileSize* deve ser igual a 0 (zero).

O método realiza basicamente 4 operações:

1. Decodifica o nome do arquivo utilizando o padrão definido pelo protocolo P2PWS.
2. Verifica se o arquivo está dentro da pasta compartilhada da aplicação.
3. Lê os próximos 1024 bytes do arquivo a partir do valor denotado por *lastFileSize*. Caso a quantidade restante de bytes seja inferior a 1024, retorna esta quantidade restante, setando o status do download para END.
4. Retorna um objeto FilePart contendo o status do download e o array de bytes.

O retorno do método é um objeto do tipo FilePart (figura 3.8). Este objeto carrega consigo uma informação de status – representada pelo campo *status*, e um array de bytes contendo o pedaço do arquivo solicitado – representado pelo campo *part*.

O campo status é uma enumeração cujos valores são:

PART Indica que este pedaço é apenas uma parte do arquivo e que o download deve continuar.

END Indica que este é o último pedaço do arquivo, isto é, o fim do download.

NOT_FOUND Indica que o arquivo solicitado não foi encontrado.

DENY Indica que o arquivo solicitado não está dentro da pasta compartilhada da aplicação.

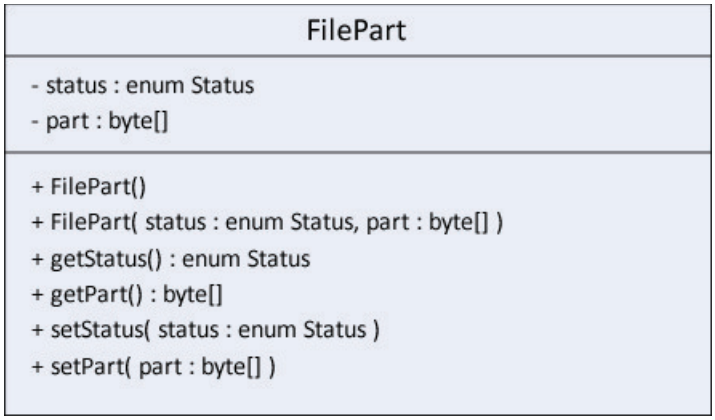

Figura 3.8: Passos do método query em pseudo-código descritivo

SERVERERROR Indica que algum erro ocorreu no servidor.

Os três últimos status (NOT_FOUND, DENY e SERVERERROR) são mensagens de erros e o array de bytes retornado no objeto FilePart é nulo. Apenas quando o status do download é PART ou END que o array de bytes será preenchido.

O método sempre retorna blocos de 1024 bytes do arquivo e somente na última parte do mesmo (que receberá o status END) poderá baixar menos que 1024 bytes, correspondendo à quantidade de bytes restantes no arquivo. As iterações deste e de todos os outros métodos da rede P2PWS ficarão mais claras na seção que segue.

3.2.4.1 Interações do método Download

Suponha que o nó X deseja baixar um recurso existente no nó A chamado "temp.txt". Então ele chama o método *download* para o nó A passando como parâmetros o path relativo do arquivo desejado no servidor (temp.txt) e a última porção de bytes baixados, que no início do download é zero, como mostra a figura 3.9. O nó A abre o arquivo "temp.txt" e retorna para o nó X um objeto FilePart, contendo a mensagem de status "PART" (indicando que o download deve continuar) e um array contendo o próximo trecho de 1024 bytes do arquivo desejado (é importante notar que o método *download* sempre baixa blocos de 1024 bytes e somente na última parte do arquivo o mesmo baixa a quantidade restante de bytes). O nó X novamente chama o método *download* porém agora com a última porção de bytes baixados equivalente a 1024. Novamente o nó A retorna um FilePart com a mensagem de status "PART" e com o segundo bloco de 1024 bytes do arquivo "temp.txt". O nó X novamente chama o método *download* porém agora

com a última porção de bytes baixados equivalente a 2048. O nó A verifica que esta é a última parte do arquivo e retorna um FilePart com status “END” (indicando o término do download) e um array de 256 bytes contendo a última porção do arquivo.

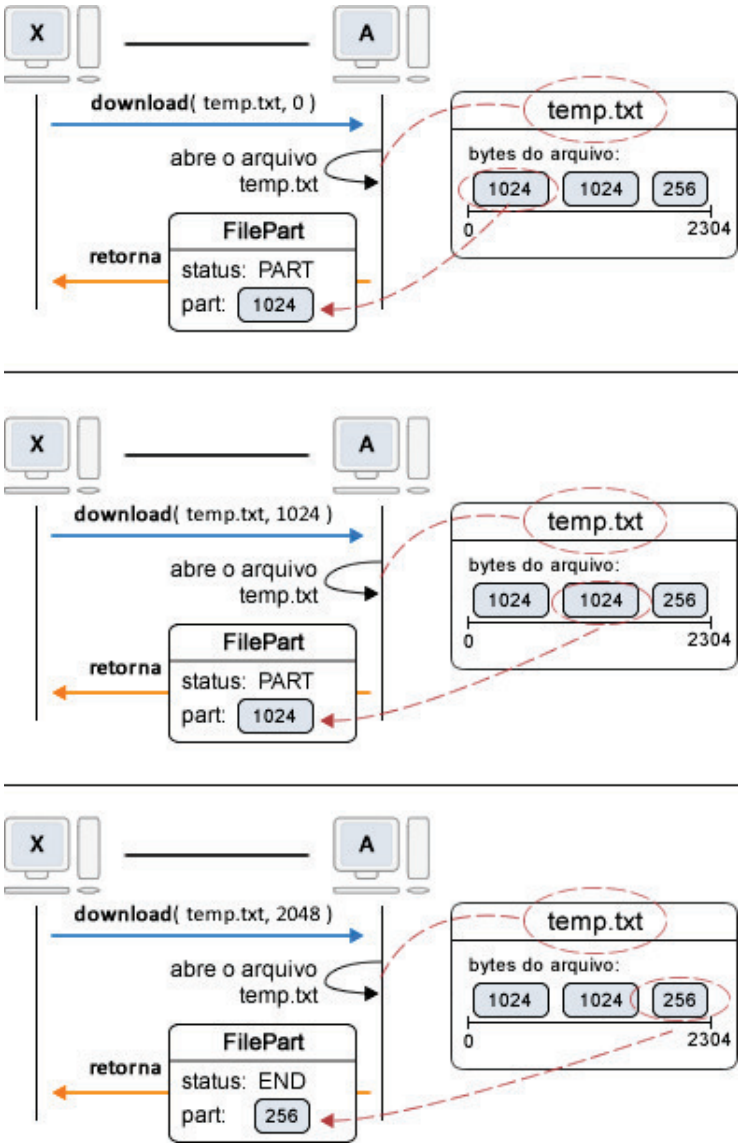

Figura 3.9: Iterações do método Download

Quando o nó solicita um arquivo que não existe no servidor, o nó solicitado retorna um objeto FilePart contendo a mensagem de status “NOT_FOUND” e o array de bytes do arquivo nulo, como mostra a figura 3.10.

Quase do mesmo modo, quando o nó solicita um arquivo que está fora da pasta compartilhada pelo nó, um objeto FilePart é retornado contendo a mensagem de status “DENY” e o array de bytes nulo, como na figura 3.11.

Por fim, qualquer erro encontrado durante o processamento do método *download*

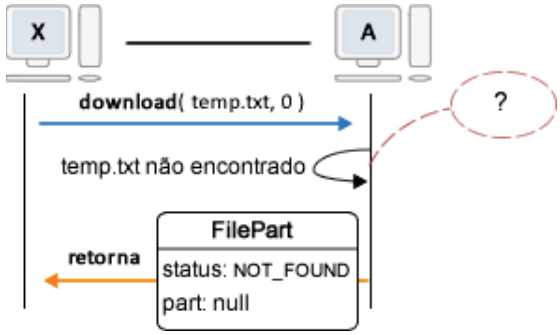

Figura 3.10: Iterações do método Download - NOT_FOUND

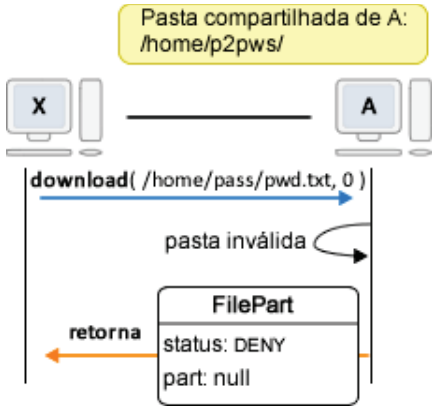

Figura 3.11: Iterações do método Download - DENY

resultará em um objeto FilePart contendo a mensagem de status "SERVERERROR" e o array de bytes nulo, como na figura 3.12.

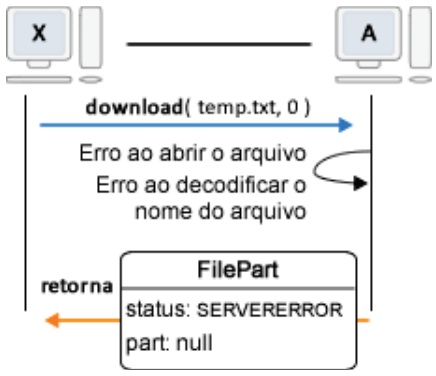

Figura 3.12: Iterações do método Download - SERVERERROR

3.3 Aplicação Cliente

Passada a etapa de definição do serviço, foi necessário desenvolver uma aplicação para demonstrar o funcionamento da rede P2PWS. Para isso, foi utilizada a linguagem Java.

O aplicativo P2PWS teria dois módulos principais, um representando o lado servidor do nó (implementando o serviço web definido pelo protocolo) e outro representando o lado cliente (fornecendo interface gráfica, administrando as informações da aplicação e realizando chamadas à serviços web).

Ao rodar pela primeira vez em uma máquina, o P2PWS exibe uma tela onde o usuário deve setar as configurações básicas do aplicativo: Pasta compartilhada, pasta de downloads, TTL das mensagens, porta utilizada para publicação do endereço wsdl (que por padrão é 6589) e – caso necessário – configurações de NAT, como mostra a figura 3.13.

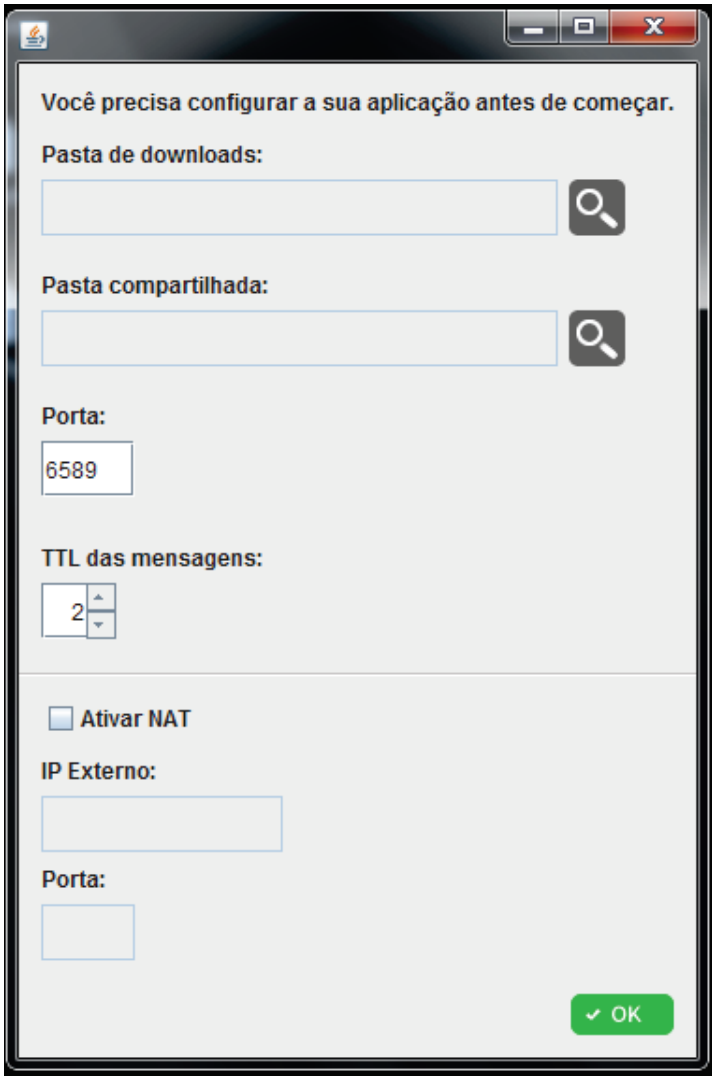

Figura 3.13: Tela de configurações iniciais do P2PWS

Passada a etapa de configuração do aplicativo, é mostrada a tela de desconexão (figura 3.14) pois o nó não possui (conhece) nenhum vizinho. Esta tela é exibida toda vez que o nó não possuir vizinhos ativos. O usuário deve inserir o endereço wsdl de um nó pertencente à rede P2PWS para realizar a chamada ao método *join* e poder integrar-se

à rede. A obtenção do endereço wsdl de um nó pertencente à rede P2PWS não faz parte da definição do protocolo.

Figura 3.14: Tela de desconexão do P2PWS

A figura 3.15 mostra a interface inicial do programa. Nela o usuário pode realizar buscas, selecionar arquivos para download e acompanhar o *status* de arquivos baixados.

A aplicação também teria que armazenar algumas informações intrisecas à arquitetura definida pelo P2PWS, como as informações de Peer, tabela de vizinhos, tabela de mensagens recebidas e algumas configurações da própria aplicação, como a pasta compartilhada, a pasta de download e o tempo de vida das mensagens geradas por este nó. Todas estas informações são armazenadas em arquivos XML e estão descritas na seção abaixo.

Figura 3.15: Tela principal do P2PWS

3.3.1 Definição de Metadados da Rede P2PWS

A aplicação possui três arquivos que contêm toda informação necessária para o funcionamento da rede. Como o P2PWS utiliza a tecnologia de serviços web e devido à sua fácil interpretação e descrição dos objetos, decidimos utilizar o padrão XML para guardar essas informações. A figura 3.16 ilustra o que é armazenado por estes arquivos para um determinado nó "A" da rede.

3.3.1.1 Definição de configurações (config.xml)

O arquivo de configurações armazena as informações de uso do aplicativo. Como pode ser visto no retângulo vermelho da figura 3.16, ele armazena as informações de Peer, o endereço da pasta de downloads (que indica onde os arquivos baixados através do P2PWS

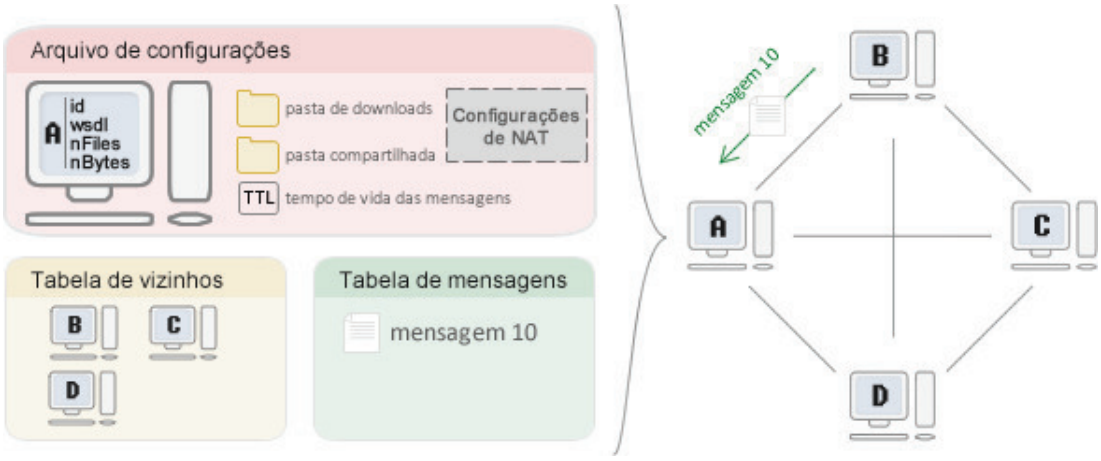

Figura 3.16: Base de metadados em XML com informações sobre a rede P2PWS

devem ser salvos), o endereço da pasta de arquivos compartilhados (que indica qual pasta está sendo compartilhada na rede), o tempo de vida das mensagens (TTL) geradas pelo nó, a porta utilizada pela aplicação e as configurações de NAT. O arquivo segue o seguinte formato ilustrado na figura 3.17.

```
<root>
    <peer>
        <id>Meu ID </id>
        <wsdl>Meu endereço WSDL</wsdl>
        <nFiles>Minha quantidade de arquivos compartilhados </nFiles>
        <nBytes>Minha Quantidade de bytes compartilhados </nBytes>
    </peer>
    <downloadsPath>Endereço da minha pasta de downloads</downloadsPath>
    <sharedPath>Endereço da minha pasta de arquivos compartilhados</sharedPath>
    <TTL>Tempo de vida das mensagens</TTL>
    <PORT>Porta da aplicação</PORT>
    <NATAtivo>Flag indicando se o endereçamento Nat está ativo</NATAtivo>
    <NATIP>Ip utilizado pelo nat</NATIP>
    <NATPORT>Porta utilizada pelo nat</NATPORT>
</root>
```

Figura 3.17: Formato do arquivo de configurações (config.xml)

3.3.1.2 Tabela de vizinhos (tabPeers.xml)

Este arquivo (retângulo amarelo na figura 3.16) armazena a tabela de vizinhos de um nó (todos os nós os quais ele conhece). O formato do arquivo segue o modelo da figura 3.18, onde cada elemento “<peer>” do xml descreve um objeto da classe Peer do P2PWS.

```
<root>
    <peer>
        <id>ID do vizinho</id>
        <wsdl>WSDL do vizinho </wsdl>
        <nFiles>Quantidade de arquivos compartilhados pelo vizinho</nFiles>
        <nBytes>Quantidade de bytes compartilhados pelo vizinho</nBytes>
    </peer>
    ...
</root>
```

Figura 3.18: Formato do arquivo “tabela de vizinhos” (tabPeers.xml)

3.3.1.3 Tabela de mensagens (messages.xml)

Este arquivo armazena os ids das mensagens recebidas por um nó. As mensagens geradas na rede e recebidas por um nó devem ser armazenadas para evitar duplicatas e reprocessamento. O arquivo segue o formato ilustrado na figura 3.19, onde cada elemento “<message>” representa o id de uma mensagem já processada pelo nó.

```
<root>
    <message>ID da mensagem recebida</message>
    ...
</root>
```

Figura 3.19: Formato do arquivo “tabela de mensagens” (message.xml)

4 Conclusão

Este trabalho mostrou que é possível o desenvolvimento de uma rede P2P "pura" utilizando a arquitetura SOA como base de comunicação. Além disso, o uso dos serviços web em sua infraestrutura possibilita a extensão de suas funcionalidades sem comprometer as funcionalidades existentes. A interoperabilidade, foco deste trabalho, também é alcançada pelo uso dos serviços web. Eles tornam possível a utilização das mesmas operações do P2PWS por aplicações em diferentes plataformas e linguagens de programação. Qualquer rede P2P existente pode facilmente incorporar um módulo cliente da rede P2PWS apenas invocando os serviços definidos pela mesma.

A tecnologia dos serviços web se mostrou excelente para a formação e estruturação da rede porém é falha quanto ao download de arquivos. Um serviço web tem um ciclo de processamento fechado, envolvendo a chamada, processamento e o retorno. Isto significa que criar uma conexão e enviar arquivos por streaming não se adequa ao modelo de negócio desta tecnologia.

A solução para a transferência de arquivos através dos serviços web foi realizar um "chunking" (baixar por partes) nos bytes do arquivo. Foi criado um método de download onde o cliente deve realizar sucessivas chamadas até baixar o arquivo por completo. Realizar o chunking dos arquivos funciona, mantém a aplicação interoperável, porém é uma forma muito lenta de transferir dados porque, para cada pedaço do arquivo a ser transferido, a plataforma tem que empacotar e desempacotar mensagens SOAPs e ainda processar as instruções do próprio método – que envolvem abertura e leitura de arquivo.

Foi observado também que aumentar o tamanho do buffer do método de download (que está definido para 1024 bytes por solicitação) aumenta a velocidade do mesmo. Por outro lado, buffers muito extensos podem estourar a memória do servidor. Um estudo mais aprofundado sobre este *tradeoff* entre o tamanho do buffer e os impactos causados na memória do servidor pode levar a uma melhoria de desempenho deste método. Outra alternativa é desenvolver um método de download adaptativo, que varie o tamanho do buffer de acordo com as condições de uso do servidor.

De todo modo, o uso de serviços web para transferência de arquivos ainda é precário.

O ideal é utilizar a arquiteutra P2PWS somente para a estruturação da rede e implementar uma maneira de transferir os arquivos via sockets. Estender a arquitetura para realizar a transferência de arquivos via sockets pode comprometer a interoperabilidade da rede porém o método de download tradicional continuará. A vantagem do P2PWS é que, por ser uma arquitetura extensível, é possível manter o funcionamento básico da rede e criar módulos adicionais, como a transferência de arquivos via sockets.

Outro problema da aplicação é a facilidade do desenvolvimento de nós egoístas (que só consomem recursos). No contexto do compartilhamento de arquivos, de forma igualitária, este é um problema comum e pode ser resolvido através do uso de protocolos de reputação, como o apresentado no trabalho (RAMOS, 2009). Por outro lado, a facilidade de desenvolver nós egoístas pode ser interessante em um contexto também egoísta, como na distribuição de conteúdo para um ambiente distribuído de TV Digital Interativa. Neste caso em particular, compartilhar recursos é do interesse dos usuários – que desejam que suas produções independentes sejam vistas, sem se importar se os outros nós vão apenas consumir. Vale ressaltar que o compartilhamento de recursos em um ambiente de TV Digital distribuido pressupõe que o problema de streaming de arquivos do P2PWS esteja resolvido.

A principal contribuição deste trabalho está na estruturação de uma rede P2P orientada a serviços, que pode ser estendida para diversas aplicações. O P2PWS possibilita a formação de uma rede de nós independentes de um servidor central, que pode ser utilizada – por exemplo – para troca de mensagens instantaneas, compartilhamento de serviços, etc.

Como trabalhos futuros, destaca-se o desenvolvimento e análise de um método de download adaptativo, o estudo de uma alternativa para o streaming de arquivos na rede, a realização de uma avaliação de desempenho da rede em experimentos em ambientes reais, comparando a rede P2PWS e a rede Gnutella v0.4 em termos de tempo de resposta e quantidade de mensagens trocadas na rede, melhorar o mecanismo de busca através da inclusão de metadados sobre o conteúdo dos arquivos, adicionar um módulo para troca de mensagens instantaneas e possibilitar o compartilhamento de serviços.

Referências

A Universally Unique IDentifier (UUID) URN Namespace. 2005. Disponível em: http://www.ietf.org/rfc/rfc4122.txt. Acessado em dezembro de 2012.

EMULE Project. Disponível em: http://www.emule-project.net. Acessado em dezembro de 2012.

ERL, T. *SOA - principles of service design*. [S.l.]: Prentice Hall, 2007. ISBN 9780132344821.

GNUTELLA Protocol Specification v0.4. Disponível em: http://www.stanford.edu/class/cs244b/gnutella_protocol_0.4.pdf. Acessado em novembro de 2012.

HTML application/x-www-form-urlencoded. 2012. Disponível em: http://www.w3.org/TR/html4/interact/forms.html#h-17.13.4.1. Acessado em dezembro de 2012.

JAVA URLEncoder Class. 2012. Disponível em: http://docs.oracle.com/javase/1.5.0/docs/api/java/net/URLEncoder.html. Acessado em dezembro de 2012.

LOPES, P. C. M. *Busca exaustiva em redes P2P*. Setembro 2010. Disponível em http://www.facom.ufms.br/gestor/titan.php?target=openFile&fileId=477. Acessado em novembro de 2012.

NETO, J. R. C. *Uma Rede de Comunicação Baseada em uma Arquitetura Orientada a Serviços e Peer-to-Peer (P2P)*. Abril 2010. Monografia. DEINF-UFMA.

NEWCOMER, E.; LOMOW, G. *Understanding SOA with Web Services (Independent Technology Guides)*. Addison-Wesley Professional, 2004.

OPENNAP: open source napster server. Disponível em: http://opennap.sourceforge.net/. Acessado em dezembro de 2012.

PAPAZOGLOU, M. P. *Service-Oriented Computing: concepts, characteristics and directions*. 2003. In: Keynote for the 4th International Conference on Web Information Systems Engineering (WISE 2003). IEEE Computer Society.

PARAMESWARAN, M.; SUSARLA, A.; WHINSTON, A. *P2P Networking: An Information-Sharing Alternative*. Julho 2001. IEEE Computer.

RAMOS, F. M. P. *Definição de uma arquitetura P2P baseada em reputação e orientada a serviços*. Abril 2009. Dissertação. DEE-UFMA.

ROCHA, J. ao et al. *Peer-to-Peer: Computação Colaborativa na Internet*. Maio 2004. Minicurso SBRC.

W3C Extensible Markup Language (XML). Disponível em: http://www.w3.org/XML/. Acessado em dezembro de 2012.

W3C Web Services Architecture Requirements. 2002. Disponível em: http://www.w3.org/TR/2002/WD-wsa-reqs-20020429#N100CB. Acessado em dezembro de 2012.

W3C Web Services Description Language (WSDL) Version 1.2. 2002. Disponível em: http://www.w3.org/TR/2002/WD-wsdl12-20020709/. Acessado em dezembro de 2012.

W3SCHOOLS Web Services. 2012. Disponível em: http://www.w3schools.com/webservices/default.asp. Acessado em dezembro de 2012.