ULBRA CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS

COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Recredenciado pela Portaria Ministerial nº 3.607 - D.O.U. nº 202 de 20/10/2005

Jan Tarik Martins Nazorek

DESENVOLVIMENTO DE UM SOFTWARE DE BACKUP DESCENTRALIZADO, UTILIZANDO A PLATAFORMA JXTA

Palmas

2013

Jan Tarik Martins Nazorek

DESENVOLVIMENTO DE UM SOFTWARE DE BACKUP DESCENTRALIZADO, UTILIZANDO A PLATAFORMA JXTA

Trabalho de Conclusão de Curso (TCC) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Profª. Mestre Madianita Bogo Marioti.

Palmas

2013

JAN TARIK MARTINS NAZOREK

DESENVOLVIMENTO DE UM SOFTWARE DE BACKUP DESCENTRALIZADO, UTILIZANDO A PLATAFORMA JXTA

Trabalho de Conclusão de Curso (TCC) elaborado e apresentado como requisito parcial para obtenção do título de bacharel em Sistemas de Informação pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. Mestre Madianita Bogo Marioti.

Aprovada em xxxxxxx de 20XX.

BANCA EXAMINADORA

_____

Prof. M.Sc. Madianita Bogo Marioti
Centro Universitário Luterano de Palmas

_____

Prof. M.Sc. Jackson Gomes de Souza
Centro Universitário Luterano de Palmas

_____

Prof. M.Sc. Edeilson Milhomem Silva
Centro Universitário Luterano de Palmas

Palmas

2013

SUMÁRIO

RESUMO

As redes de computadores têm se tornado comum entre usuários e organizações e, consequentemente, a utilização de dados digitais tem crescido. Com isso, é importante criar novas formas de armazenar esses dados com intuito de manter cópias de segurança de arquivos para evitar possíveis perdas. Assim, o objetivo deste trabalho foi desenvolver uma solução de *backup* P2P, utilizando JXTA, que possibilita o compartilhamento de arquivos disponíveis nas máquinas dos usuários. A aplicação oferece funcionalidades para o usuário enviar, buscar e recuperar seus arquivos, que são criptografados e armazenados em diversas máquinas de rede afim de criar cópias de segurança. Os arquivos enviados podem ser recuperados somente por quem possui a chave de criptografia, utilizada para garantir a privacidade dos arquivos.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

P2P – *Peer-to-Peer*

IP – *Internet Protocol*

FTP – *File Transfer Protocol*

URI –*Uniform Resource Identifier*

1 INTRODUÇÃO

Com a expansão da internet e a facilidade de acesso aos recursos computacionais e de telecomunicação, a quantidade de dados gerados por usuários também aumentou. Deste modo, continuamente tem sido propostos novos meios de atender a demanda de usuários e criar sistemas com poder computacional e capacidade de armazenamento elevado tem se tornado cada vez mais comum. Nesse contexto, utilizar sistemas distribuídos pode ser uma boa alternativa.

Coulouris (2007, p. 16) define sistema distribuído como “um sistema no qual os componentes de *hardware* ou *software*, localizados em computadores interligados em rede, se comunicam e coordenam suas ações apenas enviando mensagens entre si”. Estes componentes trabalham em conjunto para realizar ações solicitadas pelos usuários do sistema.

Um sistema distribuído pode trazer muitas vantagens, sendo que o compartilhamento de recursos é uma delas, pois oferece a capacidade de dividir o processamento entre os recursos, evitando que apenas um destes fique sobrecarregado de processos. Outro ponto positivo é a possibilidade de aumentar a capacidade do sistema, como o espaço de armazenamento, sem precisar descartar o que já está funcionando. Exemplos de sistemas distribuídos são: Cliente/Servidor; que utiliza um servidor centralizado para coordenar as ações e responder requisições enviadas pelos clientes; e P2P, que é constituído de pares que se comunicam diretamente entre si, podendo trabalhar tanto como servidor quanto como cliente.

Em uma rede P2P (*Peer-to-Peer*) os nós interligados podem compartilhar recursos como ciclos de CPU, armazenamento, arquivos etc. Muitas aplicações existentes utilizam P2P para que seus usuários possam compartilhar arquivos pessoais como músicas, vídeos, fotos etc. Além do compartilhamento de arquivos, o compartilhamento de *hardware* também é importante como, por exemplo, espaço em disco, pois outros usuários podem usar o espaço disponível de nós da rede para guardar seus dados. Baseado nisso, desenvolveu-se em criar um sistema de backup

descentralizado utilizando P2P, no qual as cópias podem ficar armazenadas em nós de rede. Esses nós representam as máquinas conectadas a rede local.

Um sistema descentralizado de *backup* guarda em vários locais uma mesma informação, tornando difícil que esta seja perdida caso ocorra falha em algum dos locais. Uma maneira de implementar um sistema desses é utilizando a ferramenta JXTA, que oferece recursos para a implementação de redes P2P e para o acesso aos recursos disponíveis dos pares da rede.

Segundo Sun Microsystems (2005, p. 7) “JXTA é um conjunto aberto de protocolos *Peer-to-Peer* (P2P) que permitem qualquer dispositivo de rede [...] se comunicar e colaborar mutuamente como pares”. Esta ferramenta vem sendo bastante utilizada em aplicações P2P, isto porque tem características que permitem a comunicação entre os pares independentemente de endereçamento de rede, protocolos físicos e plataforma de programação.

A criação de uma rede P2P descentralizada foi o objetivo do trabalho, e isto foi possível por meio do JXTA, que fornecer as funcionalidades da rede *Peer-to-Peer*. Com os métodos de rede P2P fornecidos pelo JXTA, foi implementada uma aplicação denominada *Peer Backup*. A aplicação, em conjunto com as funcionalidades da rede, fornece ao usuário métodos para realização de *backups* em rede.

Na primeira etapa do trabalho foram pesquisados assuntos referentes a sistemas distribuídos, peer-to-peer, JXTA e Backup colaborativo. Estes materiais foram utilizados para o desenvolvimento do referencial teórico, e foram utilizados como base para a execução do projeto. Durante a execução do projeto foram gerados artefatos de modelagem, e, a partir deles, foram criados cenários que mostram situações com os quais os usuários da aplicação pode se deparar. Os cenários, que englobam desde a criação da rede P2P descentralizada até a recuperação de um arquivo do usuário, serviram de base para a implementação dos métodos e das sequências de execução das ações.

A aplicação *Peer Backup* possui as características de um programa comum de cópias de arquivos, mas possibilita duplicar os arquivos em máquinas de uma rede local, com intuito de garantir a disponibilidade e garantir a privacidade utilizando chave de criptografia.

O trabalho está estruturado da seguinte forma: capítulo 1 que é referente à Introdução do projeto, capítulo 2 que apresenta o referencial teórico sobre os

assuntos 2.1 Sistemas Distribuídos, 2.2 Arquitetura *Peer-to-Peer*, 2.3 Tecnologia JXTA e 2.4 *Backup* cooperativo. No capítulo 3 é apresentada a metodologia de desenvolvimento do projeto e os materiais utilizados no desenvolvimento do projeto. O capítulo 4 é referente ao desenvolvimento da aplicação, no qual apresenta a arquitetura, modelo de dados, codificação e testes da aplicação desenvolvida. As considerações finais que é mostrada no capítulo 5, seguido pelas referências bibliográficas presentes no capitulo 6.

2 REFERENCIAL TEÓRICO

Nesta seção são abordados os principais conceitos referentes ao projeto, que são: 2.1 Sistemas Distribuídos, 2.2 Arquitetura *Peer-to-Peer*, 2.3 Tecnologia JXTA e 2.4 *Backup* cooperativo. A partir deles será possível fundamentar o desenvolvimento da aplicação.

2.1 Sistemas Distribuídos

As redes de computadores possibilitam que computadores se conectem e troquem informações, independentemente da localidade. As tecnologias de redes e sua abrangência cresceram bastante, e consequentemente, a quantidade de usuários e tipos de serviços oferecidos também. Para atender essa demanda, tem se destacado o uso de sistemas distribuídos, que têm a capacidade de atender grande quantidade de usuários e oferecer serviços usando recursos das redes de computadores.

Coulouris (2007, p.16) define sistema distribuído como “um sistema no qual os componentes de *hardware* ou *software*, localizados em computadores interligados em rede, se comunicam e coordenam suas ações apenas enviando mensagens entre si”. Isto é, processos que podem estar executando em computadores distintos trabalham em conjunto para executar determinada tarefa em comum e, também, compartilham recursos e informações.

Os sistemas distribuídos podem ser empregados em diversos tipos de aplicações como, por exemplo, uma partida online de dominó, no qual os usuários trocam mensagens com o servidor para realizar uma jogada e o servidor envia mensagem aos usuários sobre ações do jogo, como mostra a Figura 1.

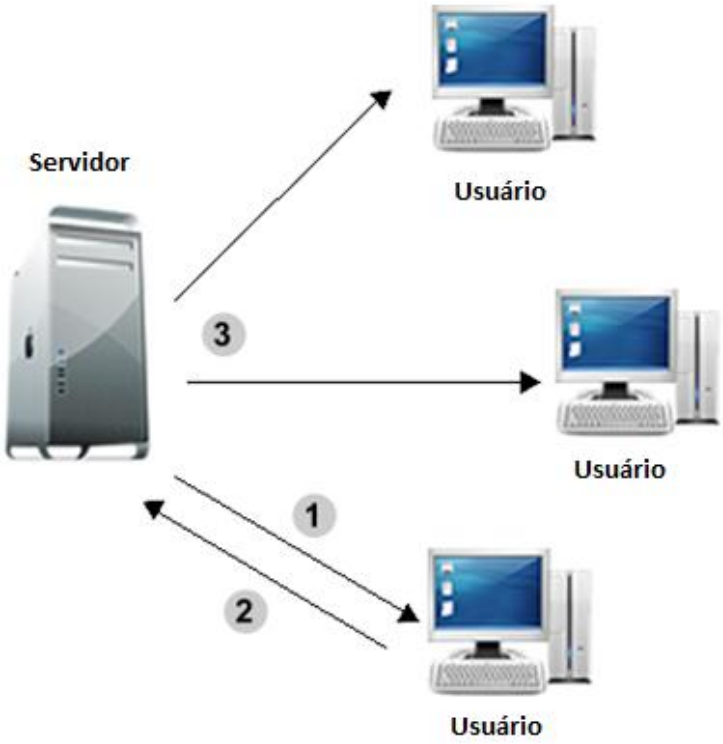


Figura 1–Troca de mensagens entre servidor e clientes

Na Figura 1, os usuários acessam a aplicação cliente que envia para o servidor mensagens de requisição. O servidor é o responsável por controlar as ações baseando-se nas mensagens que recebeu das aplicações cliente. Dessa forma, quando um usuário realiza uma jogada, o aplicativo cliente envia uma mensagem ao servidor, que replica a mensagem para os aplicativos cliente dos demais usuários para que todos os usuários tenham a mesma visão do jogo nos seus computadores.

No desenvolvimento de um sistema distribuído é necessário considerar aspectos de projeto importantes para que este funcione de forma consistente, que são: Transparência, Flexibilidade, Escalabilidade, Confiabilidade, Tolerância a falhas e Desempenho. Cada um desses aspectos é apresentado na seção seguinte.

2.1.1 Aspectos de Projeto

No projeto dos Sistemas Distribuídos um conjunto de aspectos devem ser considerados visando garantir o funcionamento adequado, para que as tarefas possam ser realizadas de forma eficiente, que seu funcionamento não seja interrompido em caso de falhas, que tenha a capacidade de acompanhar o

crescimento da demanda, entre outras características desejadas em um sistema de computação. Esses aspectos são:

- Transparência;
- Flexibilidade;
- Escalabilidade;
- Confiabilidade;
- Tolerância a falhas;
- Desempenho.

Transparência

Um sistema distribuído deve ocultar de seus usuários que os processos executados e os componentes utilizados estão separados fisicamente, mesmo quando estão separados por localidade, com sistemas operacionais diferentes etc.. Existem vários tipos de transparência, sendo que cada aplicação pode exigir conceitos diferentes (TANEMBAUM, 2007, p. 3). São elas:

- Acesso: ocultar como os dados são representados e como os recursos são acessados. Caso o usuário faça uma requisição de *download*, por exemplo, o usuário não deve perceber como este dado está armazenado no sistema;
- Localização: ocultar do usuário a localização dos recursos, sendo que recursos locais e remotos, para o usuário, são vistos da mesma forma. Por exemplo, ao acessar uma página o usuário não quer saber se ela está no Brasil ou na China, apenas necessita que página seja exibida.
- Migração: ocultar que um recurso possa ter sido movido de um local para outro. Por exemplo, se um arquivo foi movido de um servidor para outro, o sistema deve retornar as informações para o usuário sem que este perceba ou se preocupe com a nova localização;
- Performance: melhorar a capacidade (desempenho) do sistema caso seja necessário sem que o usuário perceba. Mesmo com o aumento da carga, devido ao aumento da quantidade de usuários, o sistema deve garantir que o tempo de acesso a um recurso não sofra alteração;
- Replicação: ocultar que existem cópias de um recurso. Um exemplo de replicação é um sistema que, para evitar perda de dados, faz cópias de dados que usuários enviam em outro local, mas, não informa isto ao usuário. Caso

ocorra um problema na máquina que está o documento original, a cópia pode ser utilizada, evitando que os dados do usuário sejam perdidos;

- Concorrência: ocultar do usuário que existem outros usuários compartilhando do mesmo recurso. Um exemplo de concorrência é dois usuários utilizarema mesma opção (exemplo sacar) de um caixa eletrônico, que busca no banco de dados informações na mesma tabela. O sistema deve ser percebido pelo usuário como exclusivo, como se ninguém estivesseusando os mesmos recursos que ele;
- Falha: ocultar falhas de algum recurso e ainda ocultar a recuperação dessa falha. Se um usuário tenta acessar uma imagem e o servidor tem um problema de conexão com a internet, este deve solucionar o problema automaticamente sem que o usuário perceba.

Flexibilidade

Segundo Dettenborn (2008, p. 19) “um projeto de sistemas distribuídos deve ser flexível com a capacidade de suportar alterações que não foram previstas”. Deste modo, a flexibilidade de um sistema está ligada a capacidade de realizar modificações futuras com mais facilidade. Nessas modificações podem ser incluídas funcionalidades como segurança, desempenho etc.. Assim, situações imprevistas devem ser contornadas de forma simples. Por exemplo, se o firewall utilizado é de baixa confiabilidade ele pode ser trocado por outro sem precisar alterar o sistema que é protegido por ele.

Escalabilidade

“Um sistema é descrito como escalável se permanece eficiente quando há um aumento significativo no número de recursos e no número de usuários” (COULOURIS, 2007, p.31**)**. Ou seja, é preciso que um sistema continue funcionando adequadamente caso a demanda aumente. Por exemplo, se um *site* da internet ganha novos usuários de forma progressiva e de diversos locais do mundo, deve permanecer funcionando normalmente e fornecer respostas às solicitações dos usuários em tempo adequado. Esta escalabilidade pode ser dividida em três níveis:

- Tamanho: capacidade de atender uma demanda maior de usuários.
- Distância geográfica:capacidade de atender usuários mesmo ele estando longe dos recursos.

- Facilidade de administração: mesmo com inclusão muitas organizações independentes ainda continuam fáceis à administração.

Confiabilidade

Confiabilidade "[...] refere-se à propriedade de um sistema poder funcionar continuamente sem falha" (TANEMBAUM, 2007, p.194). Um sistema confiável deve garantir a disponibilidade, a segurança dos dados transmitidos e, também, a possibilidade de se recuperar caso ocorra uma falha. Por exemplo, se um servidor que compartilha arquivos na maior parte do tempo está fora do ar, os usuários podem entender que o site não é confiável, pois existe uma dificuldade em obter qualquer informação dele dependendo do momento que em que é acessado.

Tolerância a Falhas

Sistemas são sujeitos a falhas, tanto por *software* como por *hardware*. "Quando ocorrem falhas no *hardware* ou no *software*, os programas podem produzir resultados incorretos ou podem parar antes de terem concluído a computação pretendida" (COULOURIS, 2007, p.32). Porém, mesmo quando ocorre algum problema com um de seus componentes, o sistema deve ter a capacidade de continuar funcionando, ou seja, deve ser tolerante a falhas. Algumas alternativas para prevenção de falhas são: redundância de *hardware*, que é ter cópias idênticas de *hardwares* utilizados; e recuperação por *software*, pois, caso algo falhe, o sistema pode, por exemplo, reiniciar um serviço ou direcionar os usuários para um recurso que está funcionando, evitando que o serviço pare.

Desempenho

É preciso que um sistema distribuído ofereça um desempenho equivalente ou melhor que de um sistema centralizado, fornecendo respostas rápidas, como se o sistema estivesse executando localmente. Podem ser adotadas métricas distintas para verificar o desempenho, como: tempo de resposta, utilização do sistema, quantidade consumida da capacidade da rede, entre outras.

A utilização de sistemas distribuídos é interessante em diversas situações, mas, a distribuição dos sistemas traz dificuldades. A próxima seção discorre sobre

as motivações que levam a criação dos sistemas distribuídos e a dificuldades encontradas no uso destes.

2.1.2 Motivações e dificuldades

Os sistemas distribuídos permitem que sejam criadas aplicações com a capacidade de executar tarefas que exigem alto poder de processamento, atender demanda de grande quantidade de usuários, entre outras possibilidades que se tornam motivação para o seu uso:

- **Compartilhamento de recursos**: capacidade de utilizar recursos disponíveis em maquinas da rede como *software*, *hardware* e dados.
- **Capacidade de concorrência**: caso a escala de usuários aumentar de forma desordenada, a carga pode ser distribuída entre os recursos, não sobrecarregando apenas um recurso.
- **Capacidade de expansão**: possibilidade de aumentar o poder de processamento ou/e armazenamento sem precisar se desfazer do que já está pronto.
- **Tolerância a falhas**: o sistema pode continuar funcionando mesmo quando ocorre um problema.

Por outro lado, se comparado a outros tipos de sistemas, o seu desenvolvimento é mais complexo e nem sempre as soluções fáceis são encontradas para resolver os problemas. Entre os principais problemas encontrados na sua utilização e desenvolvimento estão:

- **Complexidade de software**: implementar softwares para sistemas distribuídos são complexos, levando em consideração a sua dimensão e que seu funcionamento é não-determinístico.
- **Segurança**: como os dados são compartilhados, é preciso garantir que os dados sejam acessados somente por pessoas autorizadas.
- **Rede de comunicação**: a rede pode saturar por causa da quantidade de dados transferidos, e falhar. Além da distância física da rede entre os recursos do sistema podem afetar a velocidade, deixando o sistema lento.

Apesar das dificuldades o uso de sistemas distribuídos costuma ser uma boa solução para resolver determinados problemas, como o acesso à informação em localidades distintas. No seu desenvolvimento deve ser levada em consideração a necessidade, a dificuldade e o benefício que trará quando concluído. De toda forma existem várias soluções parao problema da segurança e das redes de comunicação.

Ao se decidir pelo desenvolvimento de um sistema distribuído, deve-se definir qual modelo será utilizado, já que existem alguns modelos, entre eles Cliente-Servidor e P2P, abordados na próxima seção. As motivações e dificuldades comentadas nessa seção estão presentes nos Sistemas Distribuídos em geral, independente do modelo.

2.1.3 Modelos de Sistemas Distribuídos

As diversas características contidas em sistemas distribuídos são empregadas de acordo com o modelo escolhido no seu desenvolvimento. Estes modelos permite que o desenvolvedor tenha conhecimento sobre o funcionamento do seu sistema. Entre os modelos existentes é possível citar: Arquitetura Cliente-Servidor e P2P.

Na arquitetura Cliente-Servidor existe uma aplicação central, denominada servidor, que é responsável por disponibilizar serviços e as aplicações acessadas pelos usuários, que são os clientes. Um servidor fica esperando por requisições de clientes, que recebem a solicitação do usuário, empacotam em uma mensagem de requisição e enviam ao servidor pela rede (TANEMBAUM, 2007, p22). Esse tipo é considerado centralizado, no qual um servidor é essencial e responsável pelo atendimento das requisições dos usuários e que uma falha neste pode gerar muitos problemas.

Vários sistemas podem ser citados como exemplo de cliente servidor, um deles é o sistema de FTP (*File Transfer Protocol*), mostrado na Figura 2.

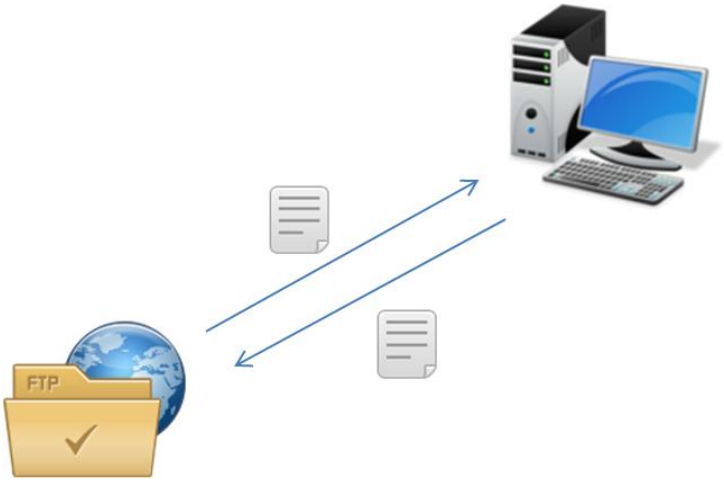


Figura 2 - Servidor FTP

Na Figura 2 é apresentado um exemplo de aplicação Cliente-Servidor, que representa o funcionamento do FTP. Quando um cliente deseja um arquivo, envia a solicitação do arquivo ao servidor. O servidor recebe a solicitação, buscar o arquivo, verifica se o usuário possui permissão para acessar, monta as mensagens com o conteúdo do arquivo solicitado e envia como resposta ao cliente.

Outro modelo de sistema distribuído é o *Peer-to-Peer*, também conhecido como P2P. Nesse modelo os nós são interconectados e têm a capacidade de atuar tanto como cliente quanto como servidor. Ainda, são capazes de se auto organizar em topologias de redes, com o objetivo de compartilhar recursos (arquivos, ciclos de CPU etc.) (LOEST, 2007, p3).

Como mostra a Figura 3 os sistemas P2P são projetados de forma que os componentes, chamados de nós, se comuniquem entre si e compartilhem recursos por troca direta, via rede, sem a necessidade de um servidor central.

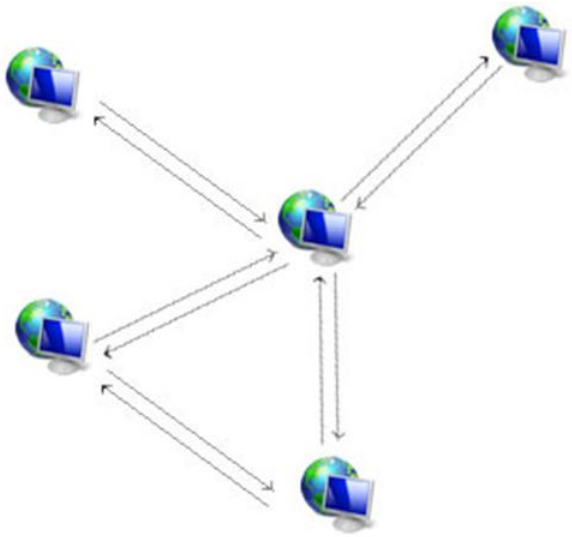


Figura 3 - Sistema distribuído P2P

Como a arquitetura P2P é o foco desse trabalho, a seção seguinte apresenta de forma mais detalhada os conceitos, exemplos e características relacionadas a essa arquitetura.

2.2 Arquitetura *Peer-to-Peer* (P2P)

Os sistemas *Peer-to-Peer* tem ganhado afinidade entre desenvolvedores de sistemas distribuídos, e isto ocorre porque as redes P2P oferecem facilidade de expansão caso esta seja necessária, além de ter um custo mais baixo, se comparada a outras arquiteturas.

Segundo Androutsellis-Theotokis e Spinellis (2004, p. 337) “Peer-to-Peer são sistemas distribuídos constituídos de nós interligados capazes de se organizar em topologias de rede com a finalidade de compartilhar recursos [...]”.

Estes nós podem trabalhar como cliente (ex. recebendo um arquivo) ou como servidor (ex. transferindo um conteúdo). Os recursos compartilhados podem ser apenas arquivos ou até mesmo ciclos de CPU, espaço em disco etc.. Na Figura 4 é mostrada a representação de uma rede P2P composta por quatro nós.

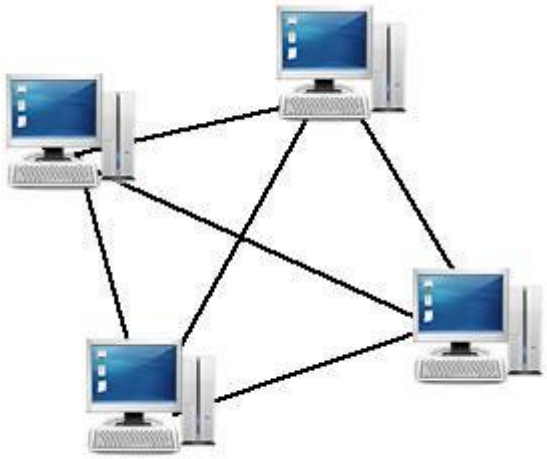


Figura 4 - Rede P2P

Na Figura 4 é mostrada a ligação entre nós em uma rede P2P, sendo que eles são equivalentes e funcionam como cliente ou como servidor. Em alguns casos, podem existir super-nós, que são pares que utilizados para centralizar informações como localização de pares, recursos etc., isto é definido na aplicação.

A arquitetura P2P pode ser empregada em diversos tipos de aplicações, como para transferência de conteúdo como fotos, vídeos, programas etc.. De acordo com a finalidade da aplicação foram definidas diversas classificações, as principais são: comunicação e colaboração, computação distribuída, suporte de serviços de internet, sistemas de banco de dados, distribuição de conteúdo. Na próxima seção é abordada as características dessas classificações.

2.2.1 Aplicações P2P

Segundo Androutsellis-Theotokis e Spinellis (2004) a arquitetura P2P está cada vez mais sendo utilizada no desenvolvimento de aplicações. Estas aplicações têm sido utilizadas para diversos fins, alguns exemplos são:

- **Comunicação e colaboração**: estão presentes nesta classificação aplicações que permitem que os usuários da rede se comuniquem, para trocar mensagens de texto, voz, imagem etc. Este tipo de aplicação tem se tornado comum entre os usuários de internet, pois existem muitos softwares que utilizam a tecnologia, como exemplo: MSN Messenger, Skype, ICQ;

- **Computação distribuída**: são sistemas que utilizam o poder de processamento dos pares da rede para executar tarefas. Este tipo de aplicação divide grandes processos em pequenas tarefas, que são distribuídas e balanceadas entre os pares e ainda utiliza o poder de processamento deles para concluir estas tarefas, deste modo o sistema ganha capacidade de processamento dependendo da quantidade de pares na rede;
- ***Streaming media***: segundo Rodrigues e Druschel (2010, p.73) “aplicações P2P mais populares são *streaming media distribution* e IPTV”. Estas aplicações *streaming* P2P tem o objetivo de utilizar a largura da banda dos pares da rede para replicar os dados de áudio e vídeo, deste modo economizando largura de banda de um servidor central;
- **Computação voluntária**: são sistema no qual os pares fornecem ciclos de CPU para terceiros. Neste caso geralmente os usuários da rede instalam uma aplicação, que recebem e processam os dados de um servidor central e retorna os resultados. Esse tipo de paradigma é bastante utilizado para cálculos científicos. Um exemplo de aplicação é o *SETI@Home* que faz a análise de sinais de rádio do espaço para detectar possíveis emissões extraterrestres (RODRIGUES, DRUSCHEL, 2010, p. 73);
- **Distribuição de conteúdo**: aplicações para distribuição de conteúdo são bastante populares entre os usuários de sistemas P2P. Estão incluídas nesta classificação aplicações que permitem o compartilhamento de arquivos (imagens, vídeos, documentos etc.) entre os usuários da rede. Exemplos de aplicações que estão nessa classificação são *Kazaa*, *Gnutella*, *Napster*.

O foco deste trabalho é a distribuição de conteúdo entre os nós de uma rede P2P, deste modo, na próxima seção é abordado o funcionamento de uma aplicação de distribuição de conteúdo.

2.2.2 Distribuição de conteúdo utilizando redes P2P

A utilização de aplicações P2P para distribuição de conteúdo tem se tornado comum entre usuários da internet. Estes serviços fornecem ao usuário meios de publicar, buscar e recuperar arquivos de outros pares ou de locais de armazenamento da rede. Os conteúdos compartilhados nessas aplicações, normalmente, incluem

arquivos de músicas, vídeos, fotos e programas. A Figura 5 apresenta uma rede P2P formada por quatro nós, que trocam conteúdos através de uma aplicação P2P.

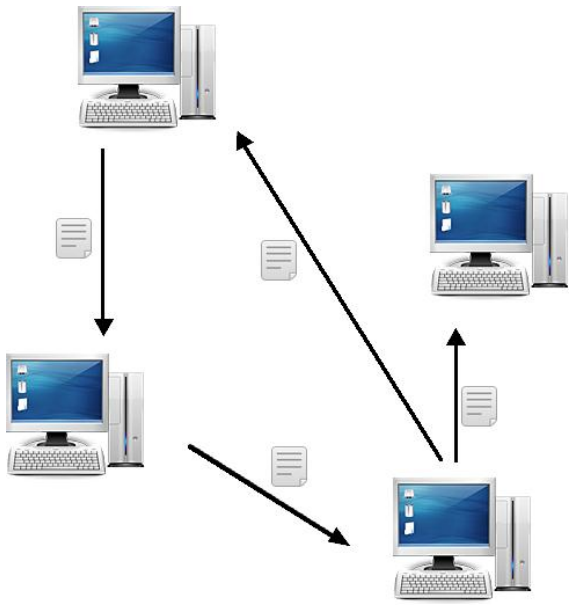


Figura 5 - Compartilhamento de conteúdo em uma rede P2P

Na Figura 5 as setas representam a troca de informações entre os pares, no qual eles podem tanto transmitir quanto receber informações. A arquitetura da rede que vai definir a ligação entre os pares, deste modo a próxima seção aborda as características das arquiteturas P2P.

2.2.3 Arquiteturas P2P de Distribuição de Conteúdo.

As redes P2P são formadas dentro de uma rede já existente (normalmente IP), e são chamadas de redes *overlay* (sobreposta). As redes *overlay* fornecem recursos como arquitetura de roteamento robusta para atingir áreas amplas, busca eficiente de dados, localização de pares próximos etc., além de características como autenticação, anonimato, escalabilidade e tolerância a falhas (LUA *etal*, 2004, p1).

As redes *overlay* podem ser diferenciadas por sua organização, na qual possuem regras para garantir o funcionamento coerente. As redes sobrepostas podem ser: estruturadas e desestruturadas.

Estruturadas

São redes com alto controle no qual o conteúdo é colocado em locais específicos, para que consultas sejam feitas de forma mais eficiente (LUA *et al*, 2004, p. 2). A

rede define chaves únicas para os dados, organiza os pares e atribui a chave para um ponto, isto permite que sejam realizadas consultas mais eficientes. Geralmente redes estruturadas são baseadas em DHT (*Distributed Hash Tables*), no qual são utilizadas para organizar os processos (ex. localização). Segundo Vestola (2010, p. 2) DHT “são sistemas descentralizados e distribuídos de prestação de serviços de pesquisa semelhante a uma tabela *hash*”.

Em redes baseadas em DHT os pares e objetos possuem identificadores únicos, conhecidos como chave. As DHTs armazenam a chave e valor de um item nos pares da rede, no qual a chave é utilizada para identificar o objeto, e o valor é o identificador do nó que é responsável pelo item. Exemplos de arquiteturas estruturadas são: *Freenet*, *Chord*, CAN.

Desestruturadas

Nas redes P2P não estruturadas a ligação entre os pares é estabelecida de forma facultativa e não existe correlação entre um par e o conteúdo administrado por ele (LUA *et al*, 2004, p. 2). Deste modo um par pode se conectar a outro utilizando somente seu endereço físico. Os itens podem ser disponibilizados em qualquer ponto da rede e para encontrá-los é necessário utilizar a busca por inundação, que consiste em difundir a requisição entre os pares. Exemplos de arquiteturas não estruturadas são: *Napster*, *Gnutella*.

Para o desenvolvimento deste trabalho, é proposta a utilização de uma rede P2P descentralizada, deste modo, a próxima seção aborda as características desse tipo de arquitetura.

2.2.4 Arquiteturas Desestruturadas

As arquiteturas desestruturadas podem ser classificadas em: hibrida descentralizada, parcialmente centralizada e puramente descentralizada.

Hibrida descentralizada

Neste tipo de rede, existe o papel de um servidor que executa ações como registro de pares, serviços de índice (recursos disponíveis), registro de grupos etc. (VITHOFT, 2009, p. 10). Os pares da rede armazenam o conteúdo e compartilham

com a rede. Na Figura 6 é apresentado o funcionamento de uma rede hibrida descentralizada.

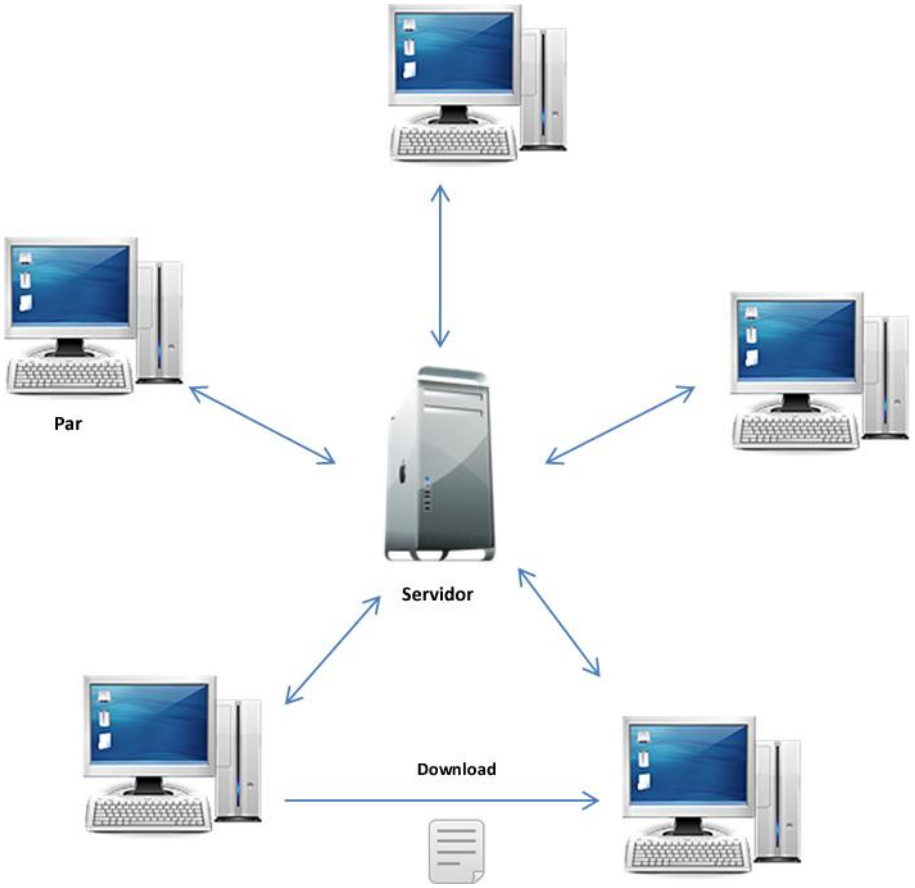


Figura 6 - Rede P2P Hibrida

Os clientes trocam informações diretamente, porém existe intermédio do servidor em casos de buscas, inicialização, publicação de recurso, conexão com a rede etc.. Como é mostrado na Figura 6, um par realiza uma consulta no servidor, que retorna a informação de localização de determinado dado. Quando obtida a informação sobre a localização, o par se conecta ao outro e realiza a transferência das informações solicitadas. Exemplos de arquiteturas híbridas são: *Napster* e *Publios*.

Parcialmente centralizada

Esta arquitetura é caracterizada pela existência de super-nós, que são definidos de acordo com as características definidas pela rede (ANDROUTSELLIS-THEOTOKIS, SPINELLIS, 2004, p. 346). Os super-nós possuem funções semelhantes à de um servidor, como exemplo indexação, *cache* e *proxy* para as requisições. Caso um

super-nó falhe, a rede possui características para se auto organizar e definir um novo. A Figura 7 mostra um exemplo de rede parcialmente centralizada.

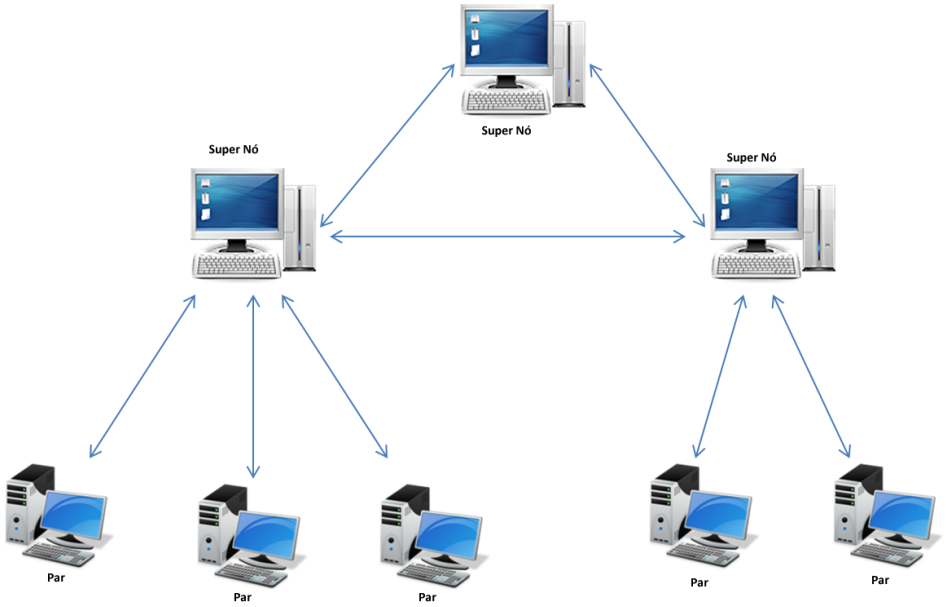


Figura 7 - Rede P2P parcialmente centralizada

Os pares comuns ingressam e publicam recursos na rede com auxilio dos super-nós. Como é mostrado na Figura 7, os super-nós são interconectados, e em conjunto com os pares comuns criam redes locais para realizar o processamento de requisições de busca. Caso os pedidos não sejam encontrados na rede local, são repassados para o super-nó referente, que realiza o roteamento da requisição entre os outros. Exemplo de arquitetura parcialmente centralizada é o *Kazaa.*

Puramente descentralizada

Esta arquitetura é caracterizada por não existir o papel de um servidor para centralizar as ações da rede, todos os pares trabalham como servidor e cliente (ANDROUTSELLIS-THEOTOKIS, SPINELLIS, 2004, p. 345). Os pares se conectam diretamente, e não necessariamente precisam conhecer todos os outros. A Figura 8 mostra um exemplo de estrutura de uma rede puramente descentralizada.

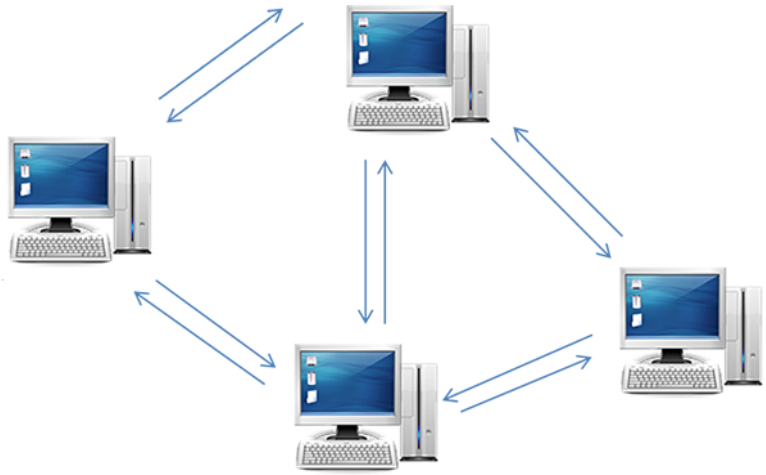


Figura 8 - Rede P2P descentralizada

Na rede P2P mostrada na Figura 8 não existe um centralizador de ações para comandar as requisições dos usuários, desta forma os pares da rede precisam se comunicam diretamente para trocar informações.

Como o objetivo do trabalho é a implementação de uma aplicação P2P descentralizada, esse tipo de arquitetura é abordada na próxima seção.

2.2.5 Arquitetura Descentralizada

Em uma rede P2P descentralizada não se utiliza um servidor para comandar as requisições enviadas na rede (BACKX*et al*, 2002, p. 2). Desta forma, não existe pontos únicos de falha, tornando mais difícil a indisponibilidade da rede. De forma arbitrária ou diretamente os pares descobrem a rede, não necessitando do servidor para registrá-los.

Qualquer par pode iniciar a rede e distribuir a tabela de vizinhos conectados para novos membros. Caso algum dos pares se desconectar, a rede continua funcionando, pois qualquer membro da rede pode fornecer os serviços essenciais da rede.

Um exemplo de arquitetura descentraliza é a rede *Gnutella* (ANDROUTSELLIS-THEOTOKIS E SPINELLIS, 2004, p. 344), que é formada por pares conectados por meio do protocolo IP, que operam como servidor e cliente e possuem uma tabela de roteamento dinâmico. A Figura 9 mostra como um par se conecta a rede.

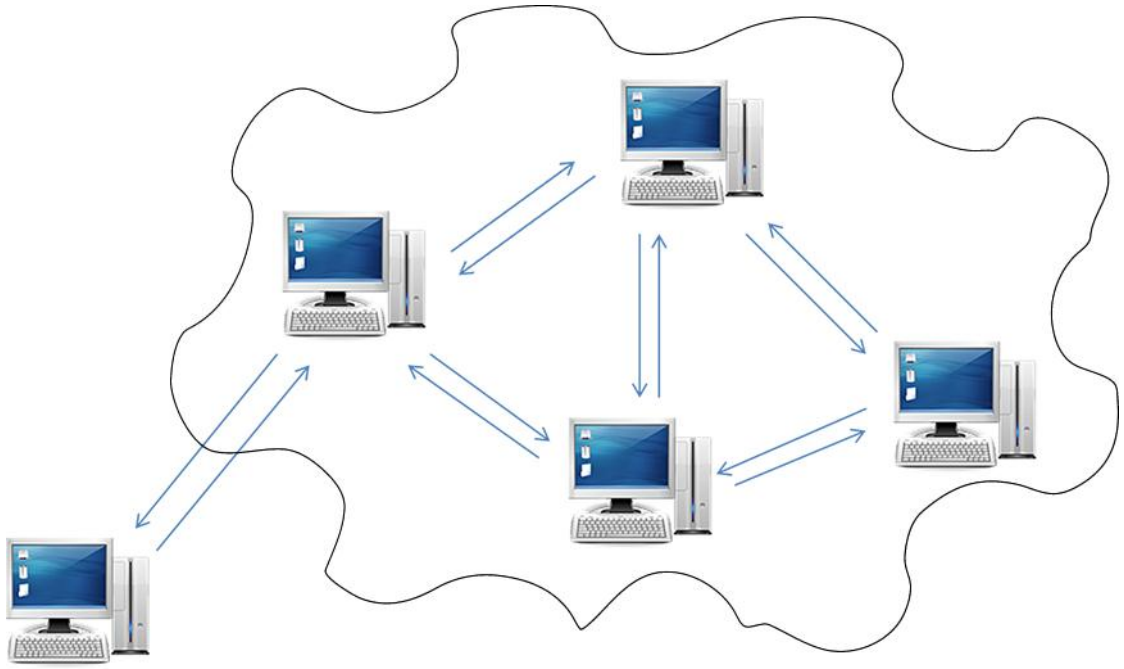


Figura 9 - Inclusão de novo membro na rede

Na rede descentralizada mostrada na Figura 9, um novo par envia uma requisição para outro já conectado a fim de se conectar na rede, e recebe uma resposta informando se a entrada foi permitida ou não. Quando conectado, um par envia mensagens de tempos em tempos para seus vizinhos com intuito de descobrir novos pares.

As mensagens servem para que os membros possam se comunicar e trocar informações. No *Gnutella* as mensagens possuem em seu corpo atributos importantes (KLINGBERG, T., MANFREDI, R., 2002, *online*), como:

- ***Message* ID**: identificado único da mensagem na rede;
- ***PayloadType***: determina o tipo de mensagem. Exemplo: *Ping*, *PongBye* etc.;
- **TTL (*Time To Live*):** controla a propagação da mensagem. No qual é definido um valor, que é decrementado a cada par percorrido. Quando o valor chega à zero significa que no tempo especificado a consulta não retornou nenhum objeto;
- ***Hops*:** quantidade de vezes que a mensagem foi encaminhada;
- ***PayloadLength*:** tamanho de bytes do corpo da mensagem.

As mensagens têm papel importante por ser o meio de comunicação entre os pares da rede (ANDROUTSELLIS-THEOTOKIS E SPINELLIS, 2004, p. 344).

Existem mensagens de controle como, por exemplo, para ingressar na rede, verificar status etc.; e para troca de dados, que são os arquivos. São definidos quatro tipos de mensagens, que são:

- ***Ping***: pedido de um determinado host para se anunciar;
- ***Pong***: resposta a uma mensagem de *ping*.
- ***Query***: mecanismo para realizar busca de recursos disponíveis na rede;
- ***QueryHit***: resposta a uma consulta. Este tipo de mensagem retorna ao usuário a localização de dados encontrados da pesquisa correspondente.

Por não existir um centralizador de ações, as mensagens são propagadas pela rede por inundação, que consiste em distribuir a requisição entre os pares vizinhos, que difunde a requisição entre outros membros. Na Figura 10 é mostrada a troca de mensagens a fim de encontrar um arquivo.

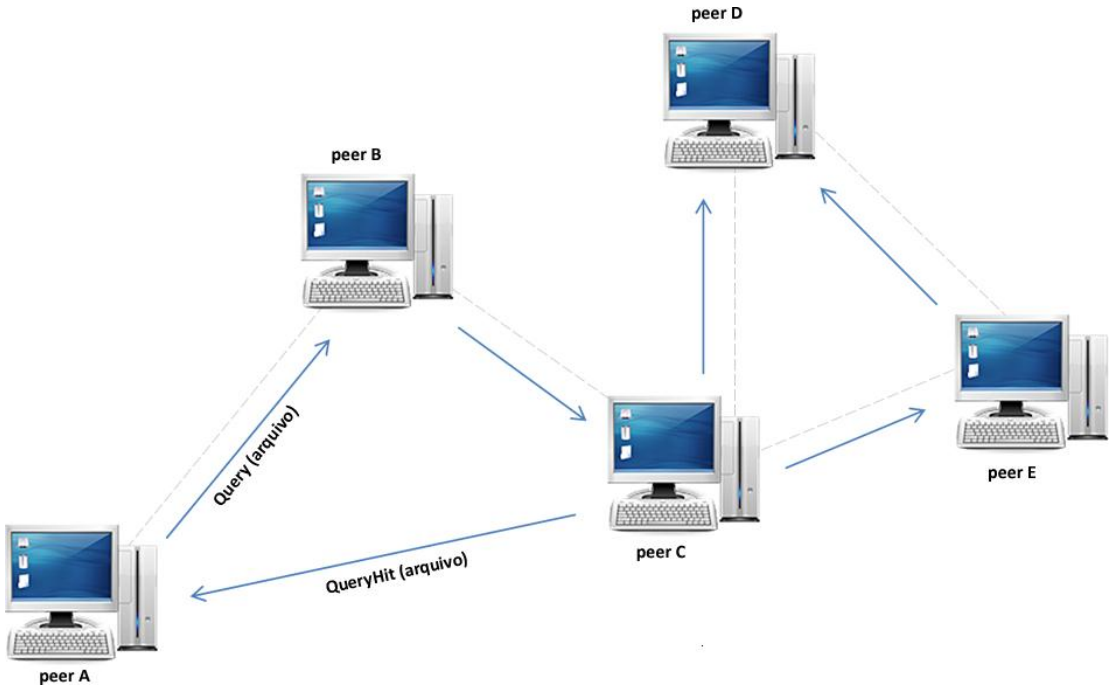


Figura 10 - Buscar um arquivo na rede descentralizada.

Para obter um determinado dado na rede, um par deve seguir alguns passos, como mostrado na Figura 10, na qual a linha tracejada representa a ligação dos pares e as setas indicam o caminho percorrido das mensagens. O "*peer* A" envia uma mensagem de Q*uery* para seu vizinho "*peer* B", que consequentemente distribui

a requisição com seu vizinho “*peer* C”, que repassa a mensagem aos vizinhos “*peer* D” e “*peer* E”. Ao encontrar o arquivo no “*peer* C”, uma mensagem de *QueryHit* é retornada ao “*peer* A”, que originou a mensagem, contendo as informações de localização do arquivo.

Em qualquer arquitetura P2P, as mensagens trocadas devem ser vistas e modificadas somente por usuários autorizados, aumentando a segurança da rede, o que é discorrido na próxima seção.

2.2.6 Segurança P2P

Para garantir a segurança de uma rede algumas características devem ser levadas em consideração (ANDROUTSELLIS-THEOTOKIS E SPINELLIS, 2004), entre elas estão:

- **Integridade e autenticidade**: garantir que as informações não possam ser modificadas por usuários não autorizados. E que membros não se passem por outros.
- **Privacidade e confidencialidade**: garantir que somente membros autorizados possam ter acesso à determinada informação.
- **Disponibilidade e persistência**: garantir que as informações e a rede sempre estejam disponíveis quando o usuário desejar ou precisar.

As características citadas tendem a criar uma sensação de segurança no usuário sobre os dados trocados na rede. Desta forma, é necessário que sejam adotados mecanismos que não permitam acesso não autorizado e a integridade das informações. Entre os mecanismos existentes estão:

- **Auto certificação dos dados;**
- **Esquema de *Shamir Secret Sharing;***
- **Retransmissão criptográfica anônima;**
- ***Public Key.***

Auto certificação dos dados

Este mecanismo permite que um par ao recuperar um arquivo verifique sua integridade. Isto é feito por meio de um *hash* criptográfico, que é calculado no

momento que o arquivo é postado e quando ele é recuperado (ANDROUTSELLIS-THEOTOKIS E SPINELLIS, 2004).

Esquema de *Shamir Secret Sharing*

Consistem em uma forma de criptografia, que gera uma chave para o arquivo (ANDROUTSELLIS-THEOTOKIS E SPINELLIS, 2004). Esta chave é dividida em várias partes e distribuída entre os pares da rede. Para recuperar o arquivo é necessário unir todas as partes, afim de obter a original.

Retransmissão criptográfica anônima

A pessoa que irá publicar o conteúdo escolhe um conjunto de “retransmissores” e envia para eles, utilizando conexões anônimas, partes criptografadas do arquivo. Os retransmissores, por sua vez, escolhem alguns pares para armazenar as partes que foram recebidas. No instante que todas as partes são armazenadas, a pessoa responsável pela publicação destrói as partes originais e anuncia o arquivo, com a respectiva lista de “retransmissores”.

Quando o cliente quer recuperar um arquivo, ele entra em contato com os “retransmissores”, que entram em contato com servidores aleatórios, para repassar a requisição para os pares que possuem o conteúdo. Por fim, esses pares decodificam o conteúdo e o enviam para o cliente.

Public Key

Mecanismo de criptografia que gerar duas chaves, chave pública e chave privada, no qual são relacionadas (MIZANI, 2005, p. 21-22). As informações criptografadas pelo mecanismo são recuperadas utilizando as duas chaves correspondentes. Em uma rede P2P a chave pública é armazena junto ao arquivo, e somente um par que possuir a chave privada pode decodificar e recuperar a informação, na Figura 11 mostra essa comunicação.

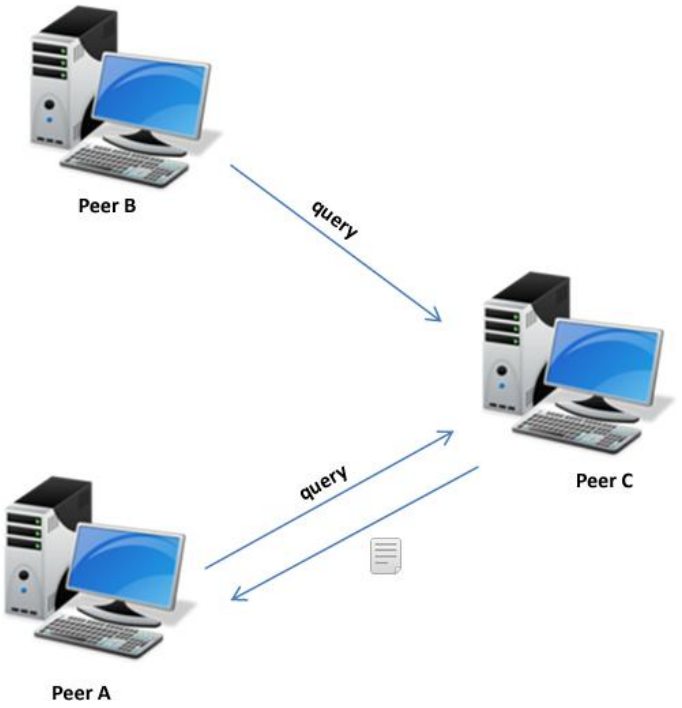


Figura 11 - Recuperando informação com chave pública

Na Figura 11 é mostrado como um par consegue recuperar uma informação, em que o "*Peer* A" possui uma chave privada de determinado objeto presente no "*Peer* C". O "*Peer* A" envia uma query requisitando o arquivo, como ele possui a chave privada correspondente a chave pública do objeto, o "*Peer* C" retorna o que foi solicitado. Já o "*Peer* B" por não possuí a chave privada necessária não recebe nenhuma informação.

Esses mecanismos de segurança fornecem confiabilidade se forem incluídos no desenvolvimento das aplicações. Existem diversas tecnologias disponíveis para se aplicações P2P, e que ainda fornece tais mecanismos, entre elas a JXTA. Que é de código aberto e vem sendo bastante utilizada em diversos projetos. Suas características são abordadas na próxima seção.

2.3 Tecnologia JXTA

O JXTA é uma tecnologia de código aberto, projetada para redes P2P, desenvolvida pela *Sun Microsystems*, com a participação de empresas, universidades, entre outros colaboradores. Segundo Gong (2001, p. 88) "a tecnologia JXTA é uma plataforma de desenvolvimento computacional e de redes que foi projetada para

resolver problemas na computação distribuída, mais especialmente para sistemas P2P".

Entre as tecnologias existentes para se desenvolver aplicações P2P, o JXTA tem ganhado afinidade entre os desenvolvedores, isto porque oferece funcionalidades que facilitam a implementação de redes P2P, que possibilitam a conexão dos pares independe da sua localização, por ser uma ferramenta independente de linguagem de programação e por funcionar em qualquer dispositivo com tecnologia digital (SUN, 2001, p. 1).

Uma rede JXTA é constituída por pares, que enviam anúncios na rede e ingressam em grupos de pares, utilizando mensagens e passando por canais virtuais denominados *Pipes,* para a formação das redes P2P(SUN, 2001, p. 4). Esses pares, anúncios, grupo de pares etc., são denominados conceitos em uma rede JXTA. Os conceitos existentes são utilizados pela tecnologia como forma de identificar uma ação ou componente da rede, que são:

- ***Peers:*** *q*ualquer dispositivo (computador, telefones, sensores etc.) que utilize qualquer protocolo JXTA, de forma que possam se comunicar entre si é considerado um *Peer* (Par). Cada par da rede funciona de forma independente e assíncrona e são identificados com um ID único. Os pares utilizam uma ou mais interfaces de rede para uso dos protocolos JXTA, anunciadas na rede como um ponto de extremidade, utilizadas para estabelecer conexões diretas entre os pares.
- ***PeerGroups:*** um grupo de pares é composto por um conjunto de pares, geralmente, com interesses em comum. O JXTA não cita normas de quando, onde e por que um grupo de pares deve ser criado. Normalmente, o objetivo é agrupar pares com interesses em comum, criar um ambiente seguro etc.
- ***Identifiers*** **(ID):** o JXTA utiliza identificadores de 128 *bits,* denominados UUID, para referir-se a entidades (par, anuncio, serviço etc). Cada entidade possui um identificador exclusivo, e para expressar esses identificadores o JXTA utiliza URI (*Uniform Resource Identifier*). As ID são apresentadas em formato texto.
- ***Advertisements****:* todos os recursos da rede são representados em forma de anúncios, como exemplo os pares, grupos de pares, serviços etc.. Os anúncios, tipo de mensagem mais trocada no JXTA, são

metadados utilizados para representar recursos. Ele pode descrever a disponibilidade (vida útil) de um recurso na rede, e pode ser republicado para aumentar a vida útil de um recurso (SUN, 2007, p. x).

- ***Messages***: são informações transmitidas por canais virtuais entre os pares, que podem armazenar qualquer tipo de dado e são transmitidas em formato XML ou em representações binárias. As mensagens ainda contêm informações de roteamento, direcionamento e a sequência de pares a ser percorrida até chegar ao destinatário.
- ***Pipes***: são canais virtuais de comunicação unidirecionais e assíncronos que conectam dois ou mais pares da rede, que possibilitam a troca de mensagens independente da sua localização, topologia e protocolo de rede. Os p*ipes* podem ser do tipo *Receiver* (recebe dados) ou *Transmitter* (transmissor de dados) (SUN, 2007, p. xi) e oferecem dois tipos de comunicação:
    - Ponto a ponto: conecta dois pontos da rede, um receptor e um transmissor de dados.
    - Propagação: faz a conexão de um ponto transmissor a vários outros pontos receptores. Todas as mensagens enviadas de um ponto transmissor são enviadas aos pontos receptores.
- ***Network Services***: serviços de rede podem ser vistos como um conjunto de operações disponibilizadas a serem utilizadas via rede. Os pares trabalham em conjunto para publicar, descobrir e invocar serviços de redes. Estes serviços são executados por um par da rede por um grupo de pares.

O JXTA se diferencia das outras tecnologias existentes por oferecer recursos que permitem que os pares conectados possam, facilmente: se localizar, se comunicar e fornecer serviços aos outros pares. Isso se deve a sua arquitetura e aos seus protocolos, que são descritos nas próximas seções.

2.3.1 Arquitetura JXTA

A arquitetura da JXTA é dividida em três camadas, que são (2002, p. 3):

- JXTA *Core*: é comparada ao *kernel* de um sistema operacional, abriga as funcionalidades primárias de uma rede P2P, entre elas comunicação (criação de pares, descoberta, gestão de comunicação, roteamento etc.) e segurança.
- JXTA *Services*: esta camada do JXTA é composta por itens que não são obrigatórios em uma rede P2P, mas, que são desejáveis. Entre as funcionalidades disponíveis nesta camada estão busca, indexação, sistema de armazenamento, compartilhamento de recursos, tradução de protocolo, autenticação etc.. Esta camada oferece recursos necessários para a colaboração entre os pares, independente da plataforma.
- JXTA *Applications*: abriga as aplicações que utilizam a tecnologia como os mensageiros instantâneos, compartilhadores de arquivos etc.. Essas aplicações utilizam as funcionalidades das outras camadas.

A Figura 12 apresenta as três camadas do JXTA, que abrigam os protocolos e as funcionalidades necessárias para o funcionamento da rede P2P.

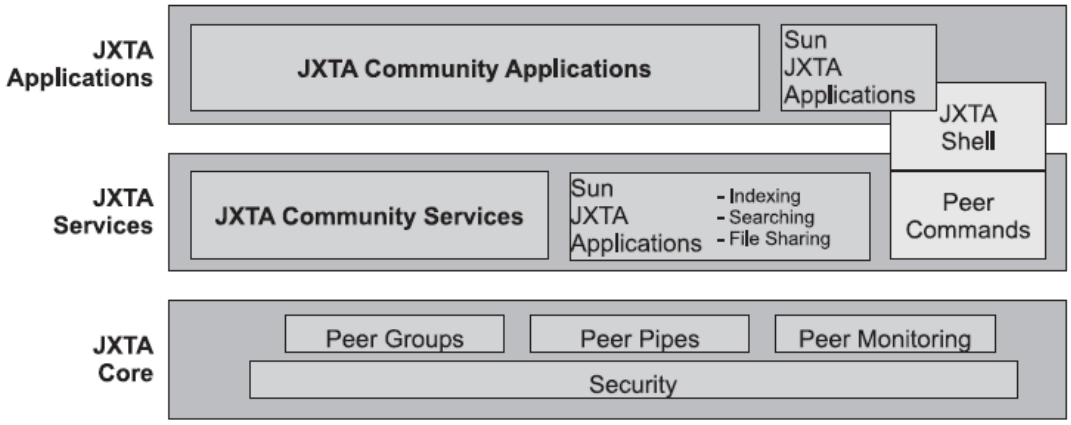


Figura 12 - Camadas JXTA (MAINBAUM, MUNDT. 2002, p. 3)

Na Figura 12 são apresentadas as camadas da arquitetura do JXTA, na qual as funcionalidades básicas para o funcionamento de uma rede P2P estão presentes na camada JXTA Core e nas outras duas camadas, JXTA *Services* e JXTA *Applications,* estão presentes os protocolos específicos de cada aplicação, que são apresentados na próxima seção.

2.3.2 Protocolos JXTA

O JXTA oferece ao desenvolvedor um conjunto de protocolos que possibilitam aos pares da rede se auto organizar e se autoconfigurar independente de sua posição na rede e sem a necessidade de um servidor para guiar as ações (SUN, 2007, p. 2).

O JXTA oferece seis protocolos, sendo que, ao se implementar uma aplicação P2P usando essa tecnologia, pelo menos um deles deve ser implementado, que são eles:

- *Peer Endpoint Protocol*: responsável pelo roteamento de mensagens e descobrir caminhos entre os pares para que as mensagens cheguem ao destino. Este protocolo ignora a existência de *firewalls* ou redes lógicas (NAT), quando um par é um roteador e possuí a rota para o destino, a mensagem é enviada diretamente para o destino final, sem necessidade de buscar novamente em outro roteador (GONG, 2001, p. 91);
- *Peer Resolver Protocol*: permite a um par da rede emitir consultas genéricas como busca de parese grupos de pares e, posteriormente, identificar as respostas correspondentes. Cada consulta ou resposta possui um identificador único (MAIBAUM, MUNDT, 2002, p. 4);
- *Peer Discovery Protocol*: este protocolo permite que um par encontre recursos anunciados na rede. Permite que o par encontre outros pares, grupo de pares e anúncios emitidos por estes. Pares que implementam esse protocolo podem, ainda, ajudar outros na busca de recursos na rede (SUN, 2007, p. 32);
- *Pipe Binding Protocol:* esta funcionalidade permite que sejam criados canais virtuais de comunicação entre pares. Estes canais virtuais permitem criar, abrir/resolver, fechar, excluir, enviar e receber operações (SUN, 2007, p. 44);
- *Peer Information Protocol*: permite que um par da rede saiba a situação de outro par. Além de obter informações como tráfego de carga, tempo de ativo etc.. Um par da rede não precisa, obrigatoriamente, implementar este protocolo, caso este não possua as mensagens de informações sobre o par são ignoradas (GONG, 2001, p. 91);
- *Peer Membership Protocol*: por meio deste protocolo, os pares podem se organizar e formar grupos com intuito de criar uma rede de segurança. Estes

pares utilizam este protocolo para ingressar ou sair de um grupo já existente na rede, e ainda para descobrir grupos existentes (GONG, 2001, p. 91).

A Figura 13 mostra a ligação existente entre os protocolos e as camadas que cada um compõe.

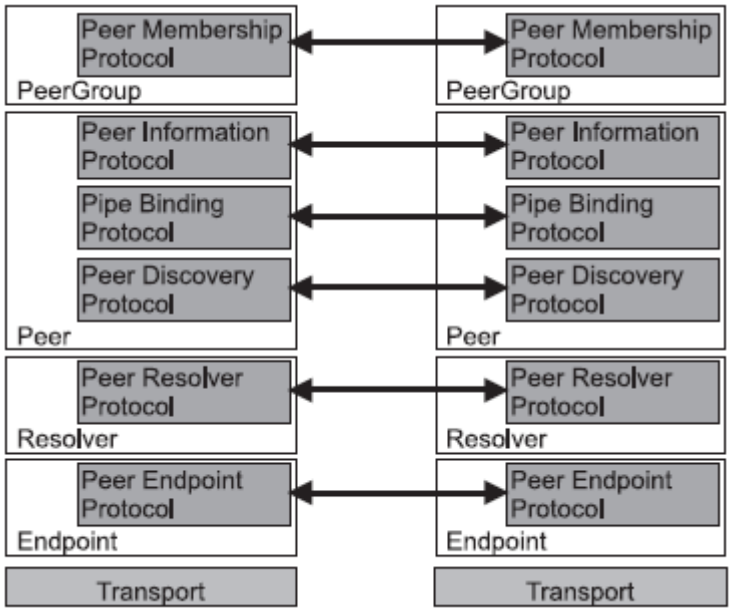


Figura 13 - Protocolos JXTA (MAIBAUM, MUNDT. 2002. p. 4)

A Figura 13 mostra os protocolos do JXTA, as setas ligam os protocolos comuns dos pares da rede. Os protocolos trabalham acima da camada de transporte, e são agrupados em quatro camadas diferentes, como mostrado na Figura 13. Na camada *Endpoit* existe o protocolo que realiza o roteamento de mensagens e descobre caminhos entre os pares. A camada *Resolver* é composta pelo protocolo que permite ao par realizar consultas na rede e identificar resposta para as buscas. A camada *Peer* é caracterizada pelos protocolos que trabalham com a descoberta de recursos na rede, a criação de canais entre os pares e ainda funcionalidades para descobrir a situação de outros pares na rede. Por fim a camada *PeerGroup*, que fornece serviços relacionados a grupo. Essas camadas servem para reunir protocolos com características em comum.

O objetivo deste trabalho é utilizar a tecnologia JXTA na criação de redes P2P para a realização de *backups* cooperativos, por isso, a próxima seção discorre sobre *backup*.

2.4 Backup Cooperativo

Com o crescimento do uso de computadores por usuários comuns e por empresas, tem-se tornado comum a utilização de dados digitais. Deste modo, a quantidade de dados armazenados em computador é cada vez maior, que podem ser documentos importantes, *softwares*, mensagens de e-mail etc.. Dessa maneira, é importante que esses dados não sejam perdidos, afim de não causar prejuízos a quem precisa das informações, e uma forma de garantir isso é a cópia de segurança, ou *backup*.

Segundo Faria (2010, p. 1) “backup (ou cópia de segurança) consiste na cópia de específicos (redundância) para serem restaurados no caso da perda dos originais”. Existem diversas formas de realizar a cópia de segurança de arquivos, que vão desde a cópia em discos até cópias em nuvem. Independente de como os dados tenham sido armazenados, os *backups* são realizados para minimizar a probabilidade de perda de dados.

“Nos serviços de backup cooperativos os recursos pertencentes a múltiplos usuários são colocados juntos como em um sistema de armazenamento distribuído para que cada um faça o backup dos dados do outro” (LOEST, 2007, p. 32). Neste caso, os usuários da rede disponibilizam espaço de armazenamento em suas máquinas para que outros usuários possam guardar seus arquivos. Assim, o usuário garante que mais cópias estejam disponíveis caso precise restaurar um arquivo perdido.

No que diz respeito às necessidades dos usuários existem alguns tipos de backup e no que diz respeito aos backups em rede podem existir duas topologias, os backups centralizados e os backups descentralizados. Essas formas de *backup* são abordadas na próxima seção.

2.4.1 Tipos de backup

Para garantir que integridade de seus arquivos, assim como as modificações feitas, os usuários tendem a realizar cópias de segurança de tempos em tempos. As cópias evitam problema com a perda de dados no futuro. Deste modo é necessário que o usuário determine estratégias para realizar estas cópias, pois de forma aleatória, os dados ou modificações, poderiam ser ignorados (JUNIOR, 2010, p. 13). Para auxiliar os usuários a criarem suas estratégias foram definidos alguns tipos de *backup*, que os mais comuns são: Completo – Full, Incremental e Diferencial.

Completo - “*Full*”

Os dados são copiados de todos os arquivos ignorando se foram modificados ou não. Esse tipo geralmente é executado em intervalos de tempo maiores do que os outros, pois o tempo de cópia pode ser muito grande dependendo do volume de dados a ser gravado. Caso seja necessária uma restauração dos dados, basta executar a última cópia gravada.

Incremental

Nesse modelo de backup são realizadas cópias dos arquivos modificados desde a última cópia completa ou cópia incremental. Na Figura 14 é apresentado como os dados de um *backup* incremental crescem durante um período de tempo.

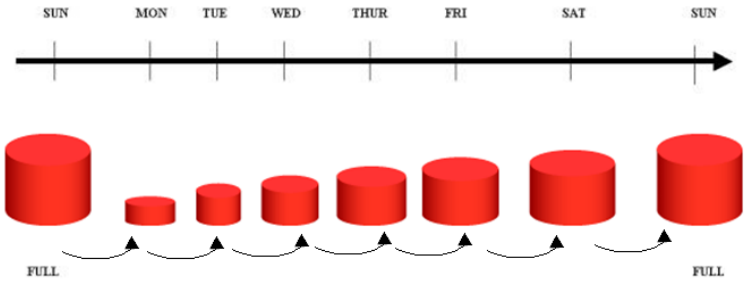


Figura 14 - Backup incremental adaptado (IBM, 2004, online)

Na Figura 14 é mostrado que uma cópia completa foi executada no domingo e os arquivos modificados ao longo da semana são adicionados ao *backup* completo, atualizando a cópia inteira. Nesse modelo, caso seja necessária uma restauração, basta executar a cópia completa.

Diferencial

Este tipo de *backup* é bem parecido com o modelo incremental, mas, os dados modificados ou criados são gravados em arquivos à parte. Na Figura 15 é mostrado o crescimento de cópias utilizando o modelo diferencial.

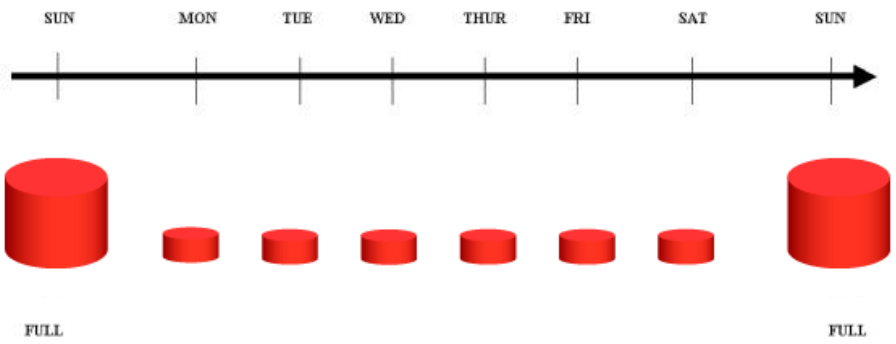


Figura 15 – *Backup* diferencial (IBM, 2004, online)

O modelo apresentado na Figura 15 mostra que foi criado um *backup* completo e ao longo do período percorrido cópias menores são criadas, essas cópias são de arquivos novos ou modificados. Nesse tipo de *backup,* caso seja necessário recuperar os itens, é preciso restaurar a cópia completa e todos os outros diferenciais criados depois dele.

Os tipos de *backup* citados são empregados de acordo com a necessidade do usuário, quantidade de dados etc.. As políticas empregadas para a criação de cópias de segurança pode usar todos os tipos de *backup*, de formas e ocasiões distintas. Por exemplo, uma empresa realiza diariamente backups diferenciais, e nos fins de semana executa um *backup* completo.

As cópias podem ser armazenada em diversos lugares, como fitas, discos externos etc. Com a disponibilidade de uma rede, os dados podem ser gravados em outros computadores. Desta forma são definidas algumas formas de backup para armazenamento em rede, e são abordados na próxima seção

2.4.2 *Backup* em rede

Utilizar computadores de uma rede para guardar arquivos pode ser uma alternativa, considerando que as informações ficam armazenadas em outros locais físicos e, caso a fonte original falhe, as cópias estarão protegidas e poderão ser recuperadas.

Em redes locais, podem ser utilizados servidores para armazenar os dados, porém, devem possuir grande capacidade de armazenamento, o que requer um alto investimento. Uma alternativa para evitar gastos é utilizar o espaço de armazenamento disponível nas máquinas da rede para guardar cópias de segurança. Deste modo existem duas topologias que podem ser empregadas aos backups em rede que são: Backup centralizado e Backup descentralizado.

Backup centralizado

Em um ambiente no qual é empregado o *backup* centralizado, os usuários utilizam uma máquina central para armazenar as cópias de segurança de seus arquivos (JUNIOR, 2010, p. 18). O computador que armazena esses backups, geralmente, são servidores com grandes capacidades de armazenamento e, normalmente, são utilizados em corporações. Na Figura 16 é mostrado um exemplo de funcionamento do backup centralizado.

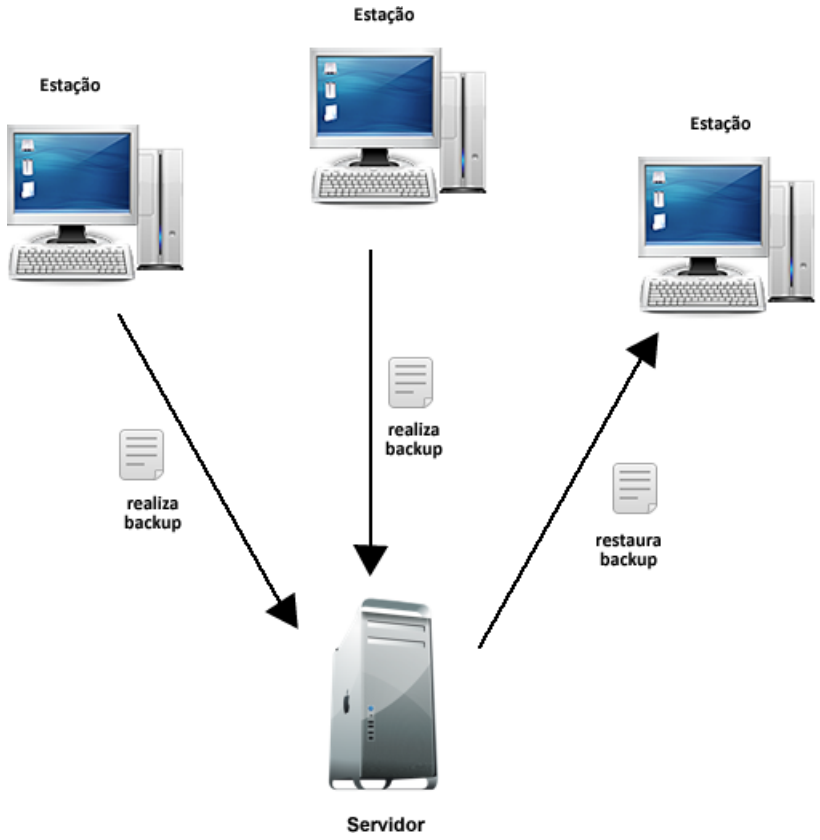


Figura 16 - Backup centralizado

Na Figura 16 é mostrada a utilização de um servidor para guardar cópias de arquivos dos usuários da rede. No qual as estações da rede efetuam cópias para o disco do servidor e, também, realizam restauração de cópias realizadas anteriormente.

Backup descentralizado

Um ambiente é descentralizado quando as máquinas da rede guardam cópias de seus arquivos em mais de um local como forma de minimizar perdas em caso de falhas (JUNIOR, 2010, p. 17). Deste modo, se ocorrer problemas em um dos locais no qual o backup foi armazenado, a restauração pode ser feita em outro local no

qual os dados foram armazenados. Um exemplo de ambiente descentralizado pode ser observado na Figura 17.

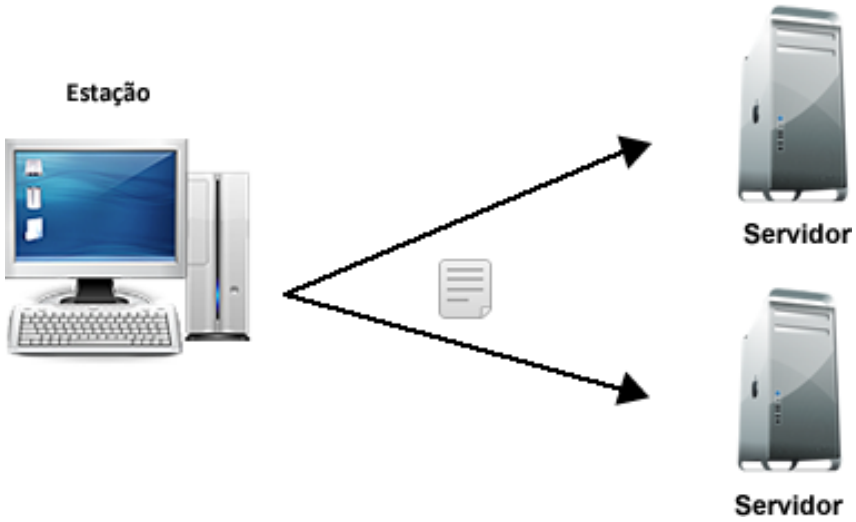


Figura 17 - Backup descentralizado

Em um ambiente que é utilizado backup descentralizado, as cópias de um mesmo arquivo são replicadas em locais diferentes, como é mostrado na Figura 17. Uma estação da rede realiza a cópia de um arquivo e este é armazenado em dois servidores diferentes. Deste modo, caso um deles venha a falhar, o outro possui uma cópia idêntica dos dados.

Os conceitos citados durante o decorrer do referencial teórico serviram como base para a execução do projeto. Os materiais utilizados durante a execução, a metodologia aplicada e os resultados dos projetos são mostrados nas seções seguintes.

3 MATERIAIS E MÉTODOS

Nesta seção são apresentados os detalhes sobre a realização do projeto, tais como materiais utilizados no decorrer do trabalho, o conjunto de hardware utilizados e a metodologia adotada no desenvolvimento da aplicação.

3.1 MATERIAIS

A primeira etapa do trabalho consistiu no estudo referente aos conceitos envolvidos no projeto. Desta forma, diversas fontes bibliográficas foram utilizadas, como livros, dissertações, artigos, teses, monografias etc.

Todas as etapas de implementação e testes foram realizados sobre o sistema operacional Windows 7, tendo como hardware um computador Intel Core2Duo E7500 2.93Mhz, 4GB de memória RAM.

No trabalho foram utilizados diversos materiais, que são divididos em duas categorias: *Hardware* e *Software*.

Hardware

Como um dos objetivos do trabalho foi a implementação de uma rede *Peer-to-Peer,* foi necessário criar um ambiente de rede com algumas máquinas para validar o funcionamento da aplicação.

Devido à dificuldade da criação de um ambiente com máquinas reais, foi decidido criar um ambiente virtual para simular uma rede de computadores. Neste ambiente foram criadas três máquinas virtuais sobre o hardware informado anteriormente.

Cada máquina virtual foi configurada com 512 Mb de memória e disco rígido de 10 Gigabytes. O sistema operacional escolhido foi o Windows XP, por não necessitar de muitos recursos de hardware para seu funcionamento.

Software

Na fase de projeto da aplicação foi utilizado o software Microsoft Visio 2010 para a criação dos diagramas de caso de uso, que descrevem funcionalidades que

necessitam da interação do usuário, e do diagrama de classes, que representa as entidades presentes na aplicação.

Na fase de desenvolvimento foram utilizados os seguintes *softwares:*

- **Java, versão 7**: linguagem utilizada para criar a aplicação de *backup.* Foi escolhida por possuir quantidade superior de referências sobre o JXTA na versão JAVA, comparado à outras linguagens de programação;
- **JXTA 2.5 versão Java**: utilizada para desenvolvimento das funcionalidades de uma rede P2P. Sendo o objetivo do trabalho utilizar esta tecnologia, e seu uso foi definido sem comparar com outras.
- **JDom 2.0**: fornece uma solução completa para a manipulação de dados XML, utilizado para acesso ao banco de dados da aplicação que é em arquivo XML. Ferramenta escolhida por ser *open source* e por fornecer as funcionalidades necessárias para manipulação de arquivos XML;
- **VMWare Player**: ferramenta gratuita utilizada para a criação das máquinas virtuais, afim de simular uma rede de computadores. Por ser uma ferramenta gratuita foi agregada ao trabalho;
- **IDE NetBeans versão 7.2.1**: ferramenta gratuita utilizada para auxiliar no desenvolvimento da interface gráfica e dos códigos;
- **Adobe Fireworks CS5**: utilizado para manipulação das imagens e ícones presentes na aplicação;
- **Microsoft Visio 2010***:* ferramenta que permite a criação de diagramas UML. Utilizado na criação de diagrama de classes e casos de uso.

As ferramentas foram utilizadas durante o processo de implementação da aplicação e a maioria foram escolhidas por serem de código aberto ou gratuitas, além da facilidade de se usar. As ferramentas auxiliaram durante o processo de execução do trabalho. A seguir é apresentada a metodologia do trabalho, que é o detalhamento das tarefas executadas com intuito de desenvolver o projeto.

3.2 Metodologia

A primeira etapa do projeto consistiu na coleta de referencial teórico sobre o domínio do projeto, que envolve Sistemas Distribuídos, P2P, JXTA e Backup cooperativo.

Paralelamente à coleta, foi realizado o estudo dos materiais coletados e a elaboração do referencial teórico.

A segunda etapa do trabalho foram executadas as fases de projeto e desenvolvimento de uma aplicação de *backup* P2P. A sequência executada para a realização do projeto é mostrada na Figura 18.

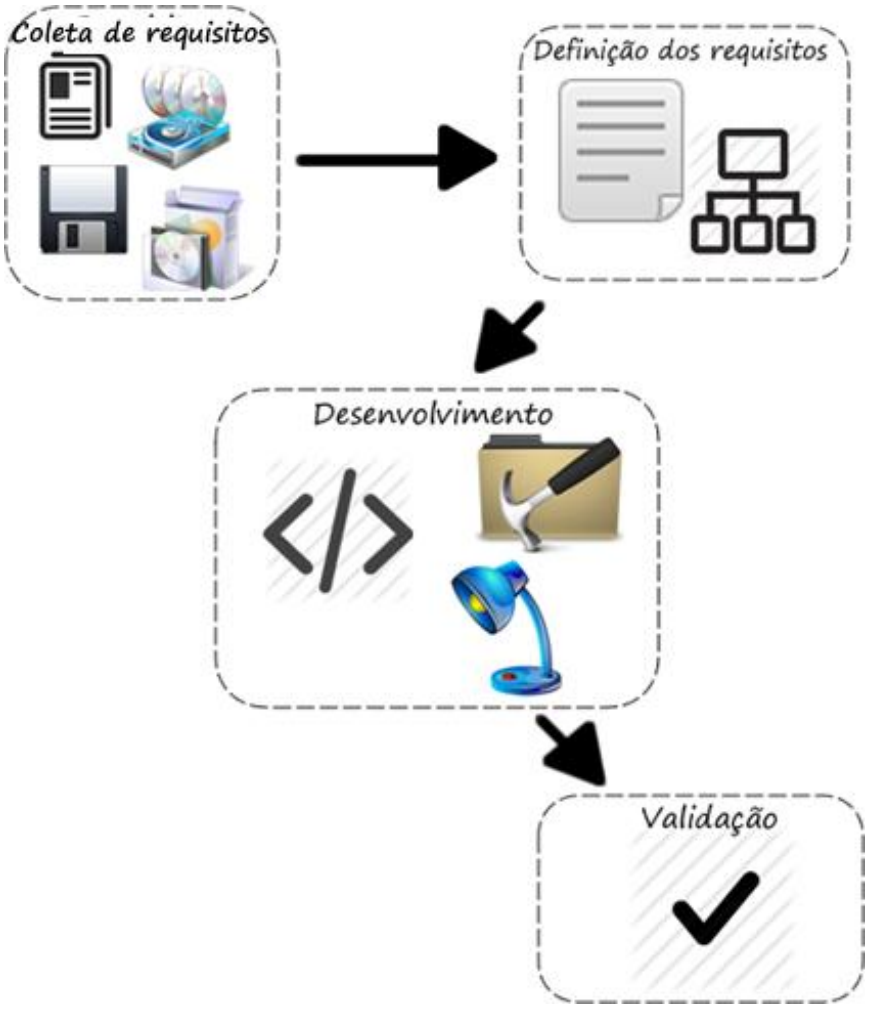


Figura 18 - Etapas de execução do projeto

Na Figura 18 são mostradas as etapas executadas durante a realização do trabalho. No início, na etapa de “Coleta de requisitos”, foi realizada uma pesquisa em sistemas de backup e compartilhamento de informações, com o intuito de verificar requisitos básicos que devem ser fornecidas ao usuário. A partir disso foram definidos os requisitos da aplicação *Peer Backup*, que é a implementação de uma aplicação de *backup* P2P.

Após a coleta de requisitos foram definidos requisitos não funcionais para a implementação da aplicação, que são:

- criação de rede P2P, de arquitetura descentralizada. Esta arquitetura foi designada por ter a característica de não precisar de um servidor para comandar as ações, sendo possível utilizar a capacidade já existente na rede local;
- envio dos arquivos completos para as máquinas da rede P2P, isto porque as cópias ficarão em computadores pessoais dos usuários que decidirem entrar na rede e no momento da restauração pode ser que nem todos os usuários estejam conectados, e caso os dados fossem separados em partes ocasionaria problema na recuperação do arquivo;
- uso de criptografia simétrica para garantir a privacidade e confidencialidade. Definiu-se chave simétrica por não ser preciso o compartilhamento da chave de criptografia, pois somente o dono dos arquivos devem recuperá-los e visualizá-los;

Por meio dos requisitos definidos, foi possível ter uma base das necessidades do software, e a partir daí foi iniciado o processo de desenvolvimento da aplicação, que consistiu em:

- criação do diagrama de classes, que representa as classes desenvolvidas na aplicação e seus respectivos relacionamentos. Por meio do diagrama foi possível definir as entidades a serem implementadas e as respostas que o sistema deve fornecer ao usuário a cada ação.
- elaboração dos cenários de uso, que são possíveis situações em que um usuário pode se deparar ao realizar uma ação via interface gráfica como o envio e recuperação de arquivos. Os cenários descritos são baseados nos casos de uso gerados, em que o usuário precisa interagir com a aplicação. Foram utilizados para definir as funcionalidades referente a manipulação de arquivos na rede P2P;
- codificação, o produto da implementação, que é aplicação propriamente dita. Consiste na implementação das funcionalidades para a criação e manutenção da rede P2P, para a manipulação do *backup* e na interface gráfica. Nesta fase foram realizadas as seguintes etapas:

- desenvolvimento da interface gráfica, a partir dela os usuários solicitam as ações, que fazem a manipulação dos arquivos e das funcionalidades da rede;
- criação do modelo de dados, que é responsável por armazenar informações dos arquivos enviados. Foi utilizado como base de dados um arquivo XML para armazenar os dados dos arquivos enviados e um arquivo de texto para armazenar o nome do usuário, dispensando a utilização de um SGBD;
- desenvolvimento das funcionalidades para a criação da rede e dos métodos utilizados pela aplicação durante o funcionamento.

Após a implementação, foi necessário realizar a validação da aplicação, com o intuito de verificar se os requisitos citados foram cumpridos. Os testes foram realizados sobre um ambiente virtual para simular uma rede de computadores, isto devido às dificuldades de se criar um ambiente real. Em um ambiente real a aplicação pode sofrer variação, como velocidade de tráfego de dados, bloqueios realizados por firewalls etc, que impacta a análise e medição de desempenho da solução.

Para a validação foram realizados os seguintes testes na aplicação, sendo que estes foram definidos:

- Teste de interface: realizado de forma manual com intuito de verificar o comportamento geral da aplicação e se o sistema oferece ao usuário o que é proposto. Nesse teste foram verificados layout, criação e autenticação da rede P2P, ações de envio, de recuperação de arquivos e configuração do sistema. Para esse teste foram utilizados usuários e arquivos fictícios;
- Teste de privacidade e autenticidade: realizado com intuito de verificar se os usuários que possuem os arquivos conseguem visualizar os dados de outros usuários, o que não deve ocorrer. Para realizar a verificação foi enviado um arquivo qualquer para *backup* e, depois, na máquina destino, foram realizadas tentativas de visualizar utilizando uma aplicação qualquer que leia o formato do arquivo enviado. A garantia da privacidade e autenticidade acontece quando um usuário da rede possui o arquivo, mas não consegue abri-lo sem a chave secreta.

Essa seção apresentou as etapas executadas para o desenvolvimento do projeto. Os resultados obtidos, os artefatos gerados e a descrição dos testes estão apresentados na próxima seção.

4 RESULTADOS E DISCUSSÕES

O presente trabalho consistiu no desenvolvimento de uma aplicação de backup P2P utilizando a plataforma JXTA, denominada *Peer Backup.* Essa aplicação fornece ao usuário funcionalidades para a criação de cópias de segurança em computadores de uma rede local, utilizando uma rede lógica P2P sobreposta a rede física. Com isso, permite que os usuários se comuniquem diretamente e que compartilhem dados e recursos como memória ram, espaço em disco, ciclos de cpu.

Os pares da rede P2P são os usuários que estão utilizando a aplicação nos computadores da rede. A comunicação entre os pares pode ser referente a busca por pares e anúncios e manipulação de arquivos. Cada par utiliza uma chave de segurança que serve para cifrar arquivos e conectar-se à aplicação.

Antes de serem enviados aos outros pares, os arquivos são cifrados com a chave do usuário e os nomes são alterados e codificados utilizando o algoritmo `base64`, com intuito de que outros usuários não tenham acesso ao nome e formato de arquivo. Os dados são enviados integralmente para os pares, desta forma, para a recuperação é preciso da chave do usuário e um membro da rede possuir o arquivo.

É importante ressaltar que as informações enviadas aos outros pares devem ser lidas e modificadas apenas por usuário que possui a chave de criptografia referente ao arquivo. Para garantir a privacidade dos dados contidos nos arquivos, é utilizada a criptografia simétrica, na qual o usuário fornece uma chave que é aplicada na cifragem dos dados.

Os critérios e requisitos da aplicação são apresentados na metodologia, e foram a base para o seu desenvolvimento. A partir dessas definições foram gerados os artefatos de software que são apresentados no decorrer desta seção na seguinte ordem:

- Arquitetura da aplicação;
- Estrutura interna da aplicação;
- Cenários possíveis no uso da aplicação;
- A aplicação *Peer Backup; e*
- Testes.

4.1 Arquitetura da aplicação *Peer Backup*

As estruturas baseadas em redes P2P se distinguem pela existência ou ausência de servidores que centralizam ações de controle da rede P2P. Nesse contexto, as redes P2P são divididas em híbridas, parcialmente centralizadas e descentralizadas, conforme foi descrito na seção 2.2.4.

A aplicação *Peer Backup* tem por objetivo utilizar recursos disponíveis nas máquinas de uma rede local, por isso foi utilizada a arquitetura descentralizada, que dispensa a necessidade de aquisição de qualquer tipo de máquina para comandar as ações da rede.

Por ser descentralizada, na *Peer Backup* os pares trabalham como servidor e/ou cliente, comunicando-se diretamente entre si e podendo iniciar uma rede P2P ou ingressar em uma já existente. A Figura 19 mostra a arquitetura da aplicação.

Figura 19 - Arquitetura Peer Backup

Na Figura 19 é mostrado vários pares conectados em uma rede P2P, sendo que podem trabalhar como cliente ou servidor. Os pares que trabalham como cliente enviam requisições para o par servidor atual. Existem mecanismos que selecionam os pares com maior capacidade computacional para servir de servidor, na aplicação caso o par que está atuando como servidor se desconecte da rede, automaticamente e de forma aleatória, outro é escolhido, evitando que a rede fique indisponível. Com isso, garante-se a tolerância a falhas.

A própria aplicação presente em cada computador fornece as funcionalidades de manipulação da rede (iniciar, autenticar, ingressar etc.) e realiza todo o processo de forma transparente ao usuário. Assim, os pares da rede são procurados e encontrados pela aplicação e não é preciso que o usuário conheça informações sobre a localização física dos demais nós como, por exemplo, endereço IP.

A implementação da aplicação foi feita em camadas, separando as funcionalidades disponíveis ao usuário das que são de uso exclusivo da aplicação. A Figura 20 mostra a divisão entre as camadas do sistema.

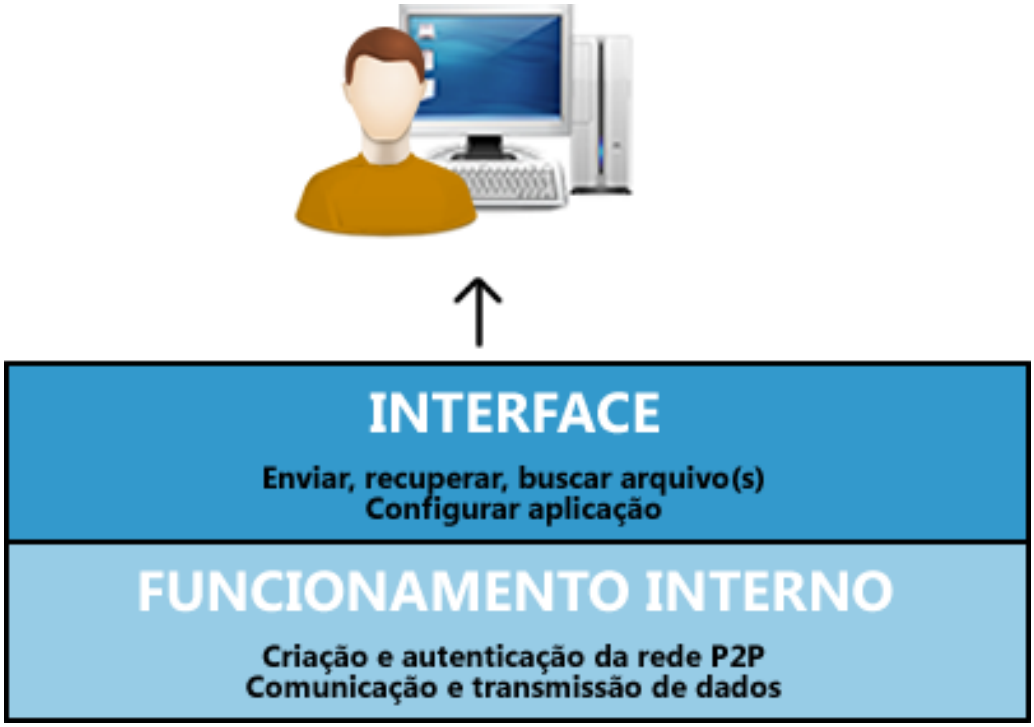


Figura 20 - Funcionamento do *Peer Backup*

A aplicação *Peer Backup* se divide em duas camadas: interface e funcionamento interno, como mostrado na Figura 20. A camada de interface fornece ao usuário uma interface gráfica, enquanto a camada de funciomento interno é responsável pelos serviços de rede e fornecer suporte à camada superior.

A camada *interface* se refere à parte visual da aplicação, através da qual o usuário pode solicitar as ações de backup, como: envio, busca e recuperação de arquivos. Essas ações utilizam métodos existentes na camada inferior.

A camada *funcionamento interno* é transparente ao usuário e fornece serviços à camada de interface, como: a criação da rede, autenticação de usuários e

comunicação entre os pares. Nessa camada são implementados os métodos de manipulação da rede, que implementam funcionalidades como verificação de integridade dos dados, localização de pares etc..

Na camada de funcionamento interno as tarefas que mantêm a rede em funcionamento são executadas por *threads*, de modo que as atividades de controle são executadas em paralelo aos outros serviços de rotina (autenticação do usuário, recuperação de arquivos etc.).

As camadas do sistema possuem dependências e trabalham em conjunto, com intuito de executar as ações solicitadas pelo usuário. Os métodos de gerenciamento de arquivo são implementados na camada de funcionamento interno, mas é por meio da interface que o usuário faz as requisições das ações implementadas pelos métodos.

Para auxiliar no desenvolvimento das funcionalidades descritas nas camadas do sistema foram gerados artefatos, que são detalhados na próxima seção.

4.2 Diagrama de classes

O diagrama de classes representa a estrutura da aplicação, como as classes e o relacionamento entre elas, além de demonstrar os atributos/propriedades e as operações do sistema referentes ao sistema. O diagrama de classes da aplicação *Peer Backup* é mostrado na Figura 21.

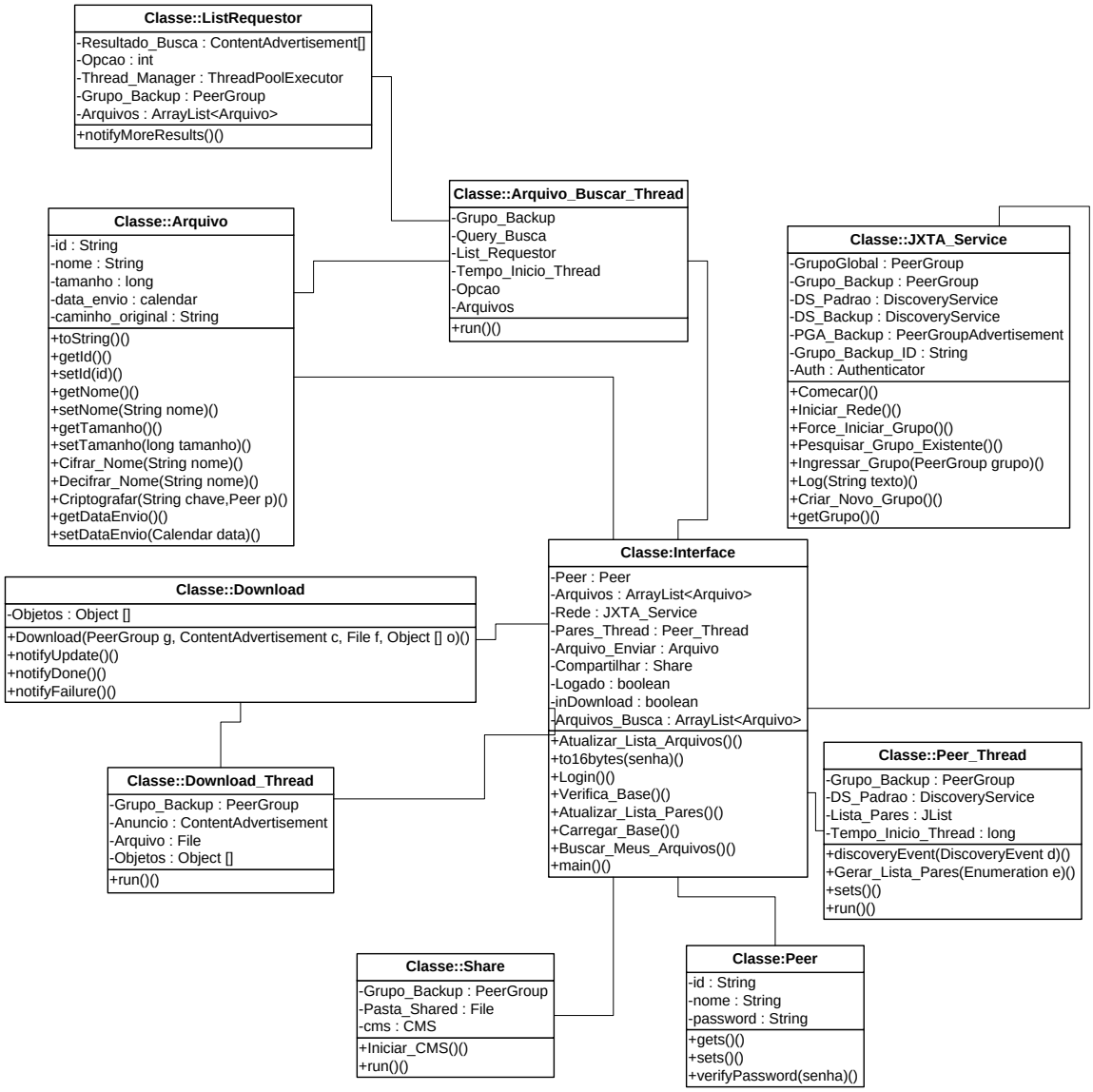


Figura 21 - Diagrama de classes da aplicação

Como mostrado no diagrama de classes da Figura 21, o sistema possui 10 classes. A classe `Interface` é referente a parte gráfica do sistema, sendo responsável por controlar o sistema e acionar métodos de outras classes. As demais classes do sistema, que fazem parte da camada de Funcionamento Interno, possuem relacionamento direto ou indireto com a classe `Interface`

O detalhamento da classe `Interface`, de seus atributos e métodos, está presente na Tabela 2.

Tabela 1 - Classe Interface

Classe: `Interface`

Responsável pela inicialização da aplicação, chamada de métodos e interface visual do sistema. Interage diretamente com o usuário.

Atributos:

- `Peer`: referente ao usuário (classe Peer) que está conectado ao sistema;
- Arquivos: lista de arquivos enviados pelo usuário;
- Rede: instância da classe `JXTA_Service`, responsável por iniciar e controlar a rede;
- `Pares_Thread`: serviço utilizado para realizar busca por pares conectados à rede;
- `Arquivo_Enviar`: referente ao arquivo que o usuário pretende enviar;
- `Compartilhar`: instância de File referente à pasta que possui os arquivos a serem compartilhados pelo usuário;
- `inDownload`: informa se existe um download em execução;
- `Arquivos_Busca`: lista de arquivos recuperados em uma varredura nos pares online.

Métodos:

- `Atualizar_Lista_Arquivos()`: atualiza lista de arquivos enviados pelo usuário;
- `to16bytes()`: transforma a senha do usuário em um objeto de 16 bytes;
- `Login()`: verifica se o usuário cadastrou uma senha para acessar o sistema, e é responsável por permitir o acesso à interface visual;
- `Verifica_Base()`: faz a verificação e criação das pastas e arquivos de dados necessários para o funcionamento do sistema;
- `Atualizar_Lista_Pares()`: realiza atualização da lista de pares conectados à rede;
- `Carregar_Base()`: carrega os arquivos de configuração referente aos arquivos e dados do usuário;
- `Buscar_Meus_Arquivos()`: inicia um serviço de busca por arquivos nos outros pares da rede;
- `Main()`: método de inicialização da aplicação.

Ao iniciar a `Interface`, são enviadas requisições para a classe `JXTA_Service`, que implementa os serviços de rede e inicia os serviços da tecnologia JXTA. Após o usuário conectar-se ao sistema utilizando sua chave secreta, a interface gráfica do sistema é exibida e são executados os processos necessários para o início da rede, autenticação de usuários e comunicação entre os pares.

Os métodos e atributos da classe `JXTA_Service` são mostrados na Tabela 3.

Tabela 2 - Classe JXTA_Service

Classe: `JXTA_Service`

Contém as funcionalidades de iniciar e controlar a rede P2P.

Atributos:

- `GrupoGlobal`: instância de `PeerGroup`, grupo P2P novo reservado a iniciar a rede;
- `Grupo_Backup`: instância `PeerGroup` referente ao grupo P2P reservado para usuários que utilizam a aplicação;
- `DS_Padrão`: instância de serviço padrão de descoberta de redes e dados utilizado para buscar serviços de rede disponíveis;
- `DS_Backup`: instância de `DescoveryService` responsável por serviço da rede *backup*, responsável pela descoberta de serviços da rede *backup*;
- `PGA_Backup`: instância de `PeerGroupAdvertisement`, responsável pelos anúncios no grupo JXTA da rede;
- `Grupo_Backup_ID`: identificação JXTA da rede;
- `Auth`: credenciais de autenticação do usuário na rede;

Métodos:

- `Comecar()`: responsável por executar os métodos de criação, busca e autenticação de grupo;
- `Iniciar_Rede()`: inicia a rede P2P;
- `Forcar_Iniciar_Grupo()`: faz requisição de criação de grupo e insere o par no grupo criado;

- `Pesquisar_Grupo_Existente()`: responsável por encontrar grupos existentes na rede P2P;
- `Ingressar_Grupo()`: inclui um par no grupo *backup*;
- `Log()`: adiciona registros de eventos ao log do usuário;
- `Criar_Novo_Grupo()`: cria novo grupo na rede P2P;
- `getGrupo()`: retorna a instância do grupo *backup*.

Os métodos e atributos apresentados na Tabela 3 são essenciais para o funcionamento da rede P2P, sendo que as ações de controle da rede são executadas a partir da classe `JXTA_Service`.

Além das classes descritas, o sistema é composto por outras classes:

- **`Peer`**: contém informações sobre o par ativo na aplicação, como senha, nome e identificação;
- **`Download`**: responsável por buscar e recuperar arquivos requisitados pelo sistema ou pelo usuário. Possuí 3 métodos para informar o status do download;
- **`Download_Thread`**: executa download de arquivos sem que seja necessário que a aplicação aguarde o seu término;
- **`Share`**: cria e anuncia serviços de compartilhamento e lista de arquivos na rede;
- **`Peer_Thread`**: executa serviço de busca de pares e adiciona à lista da interface do sistema;
- **`Arquivo`**: utilizada para representar qualquer tipo de arquivo;
- **`Arquivo_Buscar_Thread`**: responsável por iniciar serviços de busca de arquivos na rede. Recupera lista de arquivos do usuário e realiza a proliferação dos arquivos na rede;
- **`ListRequestor`**: busca por anúncios de arquivos disponíveis e recupera o arquivo para a máquina local. Possui relacionamento somente com a classe `Arquivo_Buscar_Thread`.

As classes identificadas com a terminação “thread”, são serviços executados pela aplicação, que não param ou bloqueiam a aplicação para executar uma tarefa.

Isto é, esta abordagem permite a execução de diversas tarefas em paralelo e, além disso, deixar a interface do usuário livre para uso.

A aplicação em geral, considerando até a tecnologia JXTA, geram arquivos de configuração e dados do usuário. Estes arquivos e dados criados são mostrados na seção seguinte.

4.3 Modelo de dados

A aplicação não usa um SGBD para armazenar os dados do usuário. Para a tarefa de armazenamento foram utilizados arquivos no formato XML, que contêm informações de arquivos enviados, senhas e configurações. Quando uma rede é iniciada o JXTA cria arquivos que são metadados utilizados pela tecnologia referente a configurações de usuário e grupos P2P. Estes dados são configurações de rede, anúncios espalhados pela rede, grupos criados etc.. A aplicação utiliza diretamente apenas o arquivo que possui as definições do usuário, as demais são utilizadas exclusivamente pela tecnologia. Para armazenar os dados é criada uma pasta denominada “.jxta” na raiz na qual a aplicação se encontra.

A pasta “.jxta”, armazena arquivos referentes a configuração do par e metadados com informações do grupo que o usuário ingressou, anúncios de arquivos disponíveis, informações de outros pares etc. A Figura 22 mostra os objetos presentes na pasta “.jxta”.

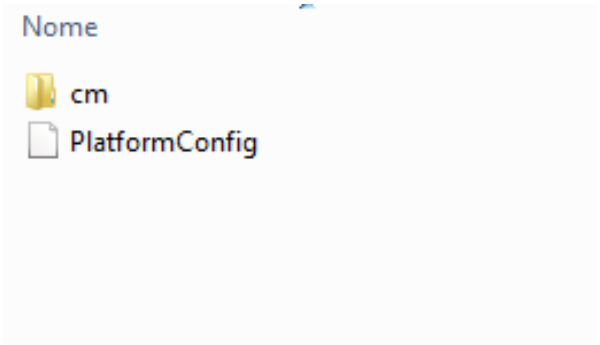


Figura 22 - Pasta .jxta

A Figura 22 mostra o conteúdo pasta “.jxta”, criada pelo JXTA: a pasta “cm” e um arquivo sem extensão de nome “PlatformConfig”. A pasta “cm” armazena os metadados, sendo que o conteúdo dos arquivos dessa pasta não pode ser lido, pois são os arquivos são criptografados pelo próprio JXTA.

O conteúdo de “PlatformConfig” é uma estrutura XML, que contém informações do usuário como identificação, nome, chave de segurança, além de configurações da plataforma, como endereço JXTA de serviços de rede, protocolos etc. A Listagem 1 mostra parte do conteúdo do arquivo “PlatformConfig”.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE jxta:CP>
3  <jxta:CP type="jxta:PlatformConfig" xmlns:jxta="http://jxta.org">
4      <PID>
5          urn:jxta:uuid-59616261646162614A78746150325033B219FB879EA24CE0A4259740D50A985D03
6      </PID>
7      <Name>
8          ghgf
9      </Name>
10     <Desc>
11         Platform Config Advertisement created by : net.jxta.impl.peergroup.AutomaticConfigurator
12     </Desc>
```

Listagem 1 - Arquivo PlatformConfig

A Listagem 1 mostra o trecho do arquivo “PlatformConfig”, que consiste em uma estrutura XML contendo informações como:

- linha 4 – elemento “PID”, que contém o endereço JXTA do par;
- linha 7 – elemento “Name”, que representa o nome do par que é utilizado pela aplicação para localizar arquivos e identificar o usuário na rede; e
- linha 11 – elemento “Desc”, que contém a descrição da configuração.

Além das informações mostradas na Listagem 1, o arquivo ainda possui outros dados:

- Endereço JXTA dos serviços fornecidos pelo par;
- Informações sobre os protocolos de transporte usado (TCP, HTTP);
- Configurações do protocolo de propagação dos anúncios; e
- Chave privada do usuário.

Os dados do arquivo “PlatformConfig” são necessários para o funcionamento da rede JXTA e são gerados pela própria tecnologia. Além dos arquivos gerados pelo JXTA, são criados arquivos pela aplicação, que são o banco de dados da aplicação e diretórios de compartilhamento e *download* de arquivos. A Figura 23 mostra as pastas criadas pela aplicação.

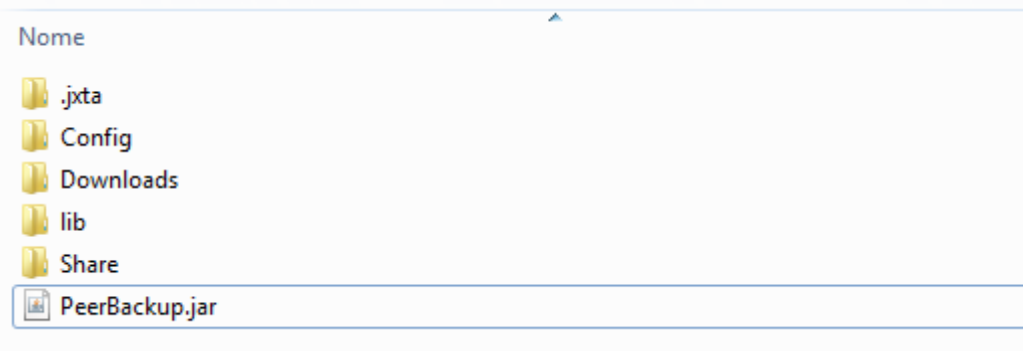


Figura 23 - Pastas *Peer Backup*

As pastas “Config”, “Downloads” e “Share”, mostradas na Figura 23, são criadas pela aplicação na sua primeira utilização. A pasta “Downloads” contém os arquivos recuperados de outros pares. A pasta “Share” contém os arquivos criptografados do usuário e de outros pares da rede e é compartilhada pelo sistema para *download* e *upload* de arquivos de outros pares. A pasta “Config” possui a base de dados do sistema, que contém informações sobre o par e arquivos enviados.

A Figura 24 mostra os arquivos criados na pasta “Config”.

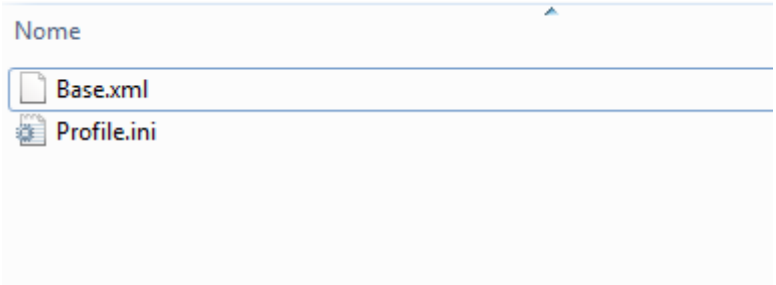


Figura 24 - Pasta “Config”

Na pasta “Config” são criados dois arquivos: “Base.xml” e “Profile.ini”. O arquivo “Profile.ini” possui apenas o nome do par criptografado com a chave do usuário, e é usado para autenticação da senha do usuário na aplicação. O arquivo “Base.xml” contém informações dos arquivos enviados pelo par, informações de data de envio, nome do arquivo etc como mostrado na Figura 25.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <arquivos>
3    <arquivo id="Sm9zZQ==#3#2__MjAtNS0yMDEzLTExNDFfX1VudGl0bGVkLTEucG5n#3#2"
4      nome="Untitled-1.png"
5      tamanho="290253"
6      data_envio="20-5-2013-1141" />
7  </arquivos>
```

Figura 25 - Arquivo "Base.xml"

O arquivo “Base.xml” contém apenas os dados referentes ao arquivo para informar ao usuário os arquivos já enviados por ele na rede. Na linha 3 é mostrado a abertura de uma chave “arquivo” com o atributo “id”, que representa o nome do arquivo cifrado, “nome”, sendo o nome original do arquivo, “tamanho” que é a quantidade de *bytes* do arquivo e “data_envio”, que é quando o arquivo foi enviado à rede.

4.4 Cenários de Uso

Ao ser utilizada a aplicação pode apresentar diversos comportamentos, que se diferem de acordo com cada ação executada pelo usuário. Esses comportamentos são representados por cenários possíveis relacionados à manipulação de arquivos na rede, sendo eles:

- **Primeiro acesso**: apresenta os passos executados pelo sistema na primeira utilização, na qual o usuário realiza um *backup* de arquivos na rede;
- **Recuperando lista de arquivos**: mostra os passos necessários para retornar a lista de arquivos enviados e armazenados em outros pares;
- **Recuperar um arquivo**: processo de *download* de um arquivo do usuário.

O primeiro cenário de um usuário é seu primeiro acesso à aplicação, que vai desde a configuração ao envio de arquivos na rede. Este cenário é mostrado na Figura 26.

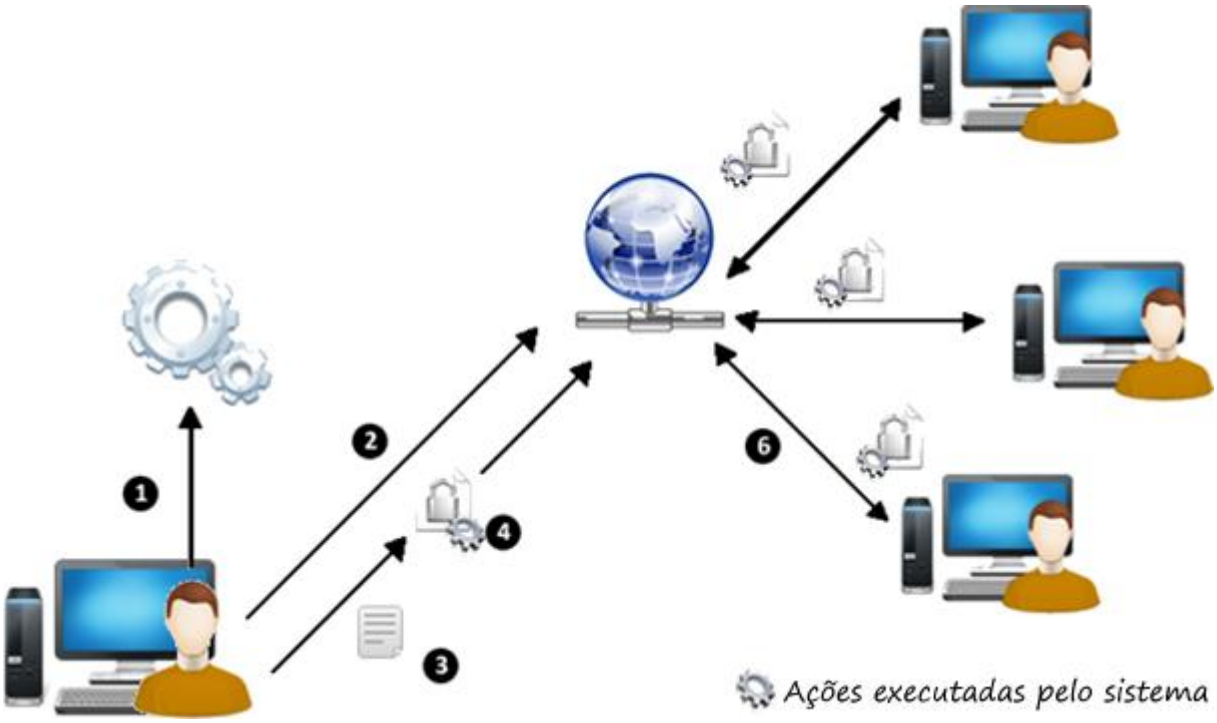


Figura 26 - Primeiro acesso

Na Figura 26 é apresentado o funcionamento da aplicação quando o usuário acessa pela primeira vez, sendo que os passos foram numerados de acordo com a ordem que a ação é executada:

- Passo 1 - configuração de nome de usuário e senha, que são necessários para cifrar e restaurar os arquivos. Ao realizar a configuração, o sistema registra o usuário na rede, caso esta esteja ativa, senão cria a rede.
- Passo 2 – o sistema realiza uma busca por redes P2P ativas; se nenhuma for encontrada, então uma é criada;
- Passo 3 - envio de arquivos do usuário para a rede. Para realizar o envio de arquivos é necessário que tenha pelo menos dois usuários na rede, caso haja, o usuário envia o arquivo, que é cifrado com a senha do usuário antes de ser enviado aos pares disponíveis. O arquivo é cifrado utilizando com a chave do usuário utilizando a tecnologia AES, sendo que apenas o proprietário do objeto precisar conhecer a chave.

O usuário precisa configurar o sistema apenas uma vez em cada máquina que utilizar, pois é criada uma base de dados contendo seu nome de usuário e seus arquivos enviados. A base de dados, que fica armazenada em arquivo no formato XML no diretório “Config” que se encontra junto com a aplicação, é acessada

quando o usuário precisa saber quais arquivos foram guardados. O arquivo possui a identificação dos arquivos enviados, nome, tamanho, data de envio etc.. Após o envio dos arquivos, o usuário pode recuperá-lo de qualquer outro par da rede, necessitando configurar apenas o *login* e senha novamente.

Na Figura 27 é mostrado o cenário de uso que representa o processo de recuperação de arquivos.

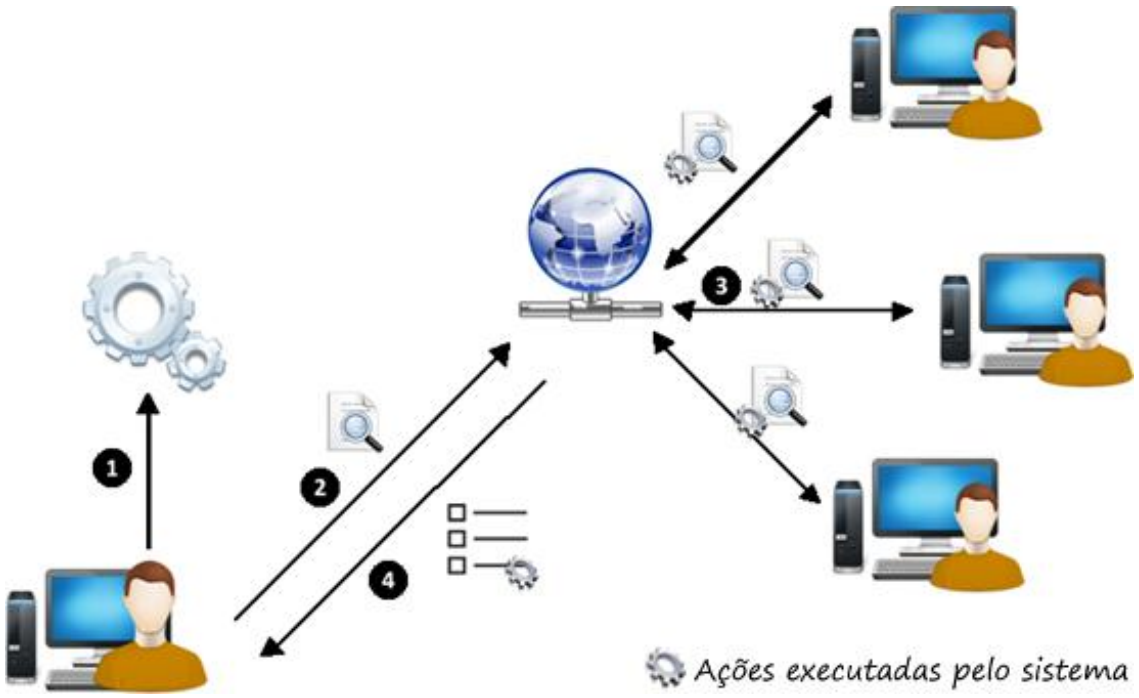


Figura 27 - Recuperando lista de arquivos

Na Figura 27 é mostrada a sequência dos procedimentos necessários para que um usuário consiga recuperar seus arquivos enviados que são:

- Passo 1 - é realizada a configuração do sistema, de forma que o usuário deverá fornecer seu usuário e senha, que devem ser as mesmas do primeiro envio de arquivos.
- Passo 2 - a aplicação envia uma requisição de busca por arquivos na rede;
- Passo 3 – os pares conectados na rede fornecem uma resposta com os arquivos disponíveis do usuário;
- Passo 4 é retornada a resposta da consulta, que é a lista de arquivos disponíveis do usuário, a lista ainda pode retornar vazia caso não seja encontrado arquivos ou não exista pares disponíveis.

Os arquivos são recuperados quando o usuário faz um requisição no sistema, que realiza a busca pelos arquivos disponíveis na rede e retorna a lista para a interface do usuário.

A Figura 28 mostra o cenário que representa o processo para realizar *download* de arquivos.

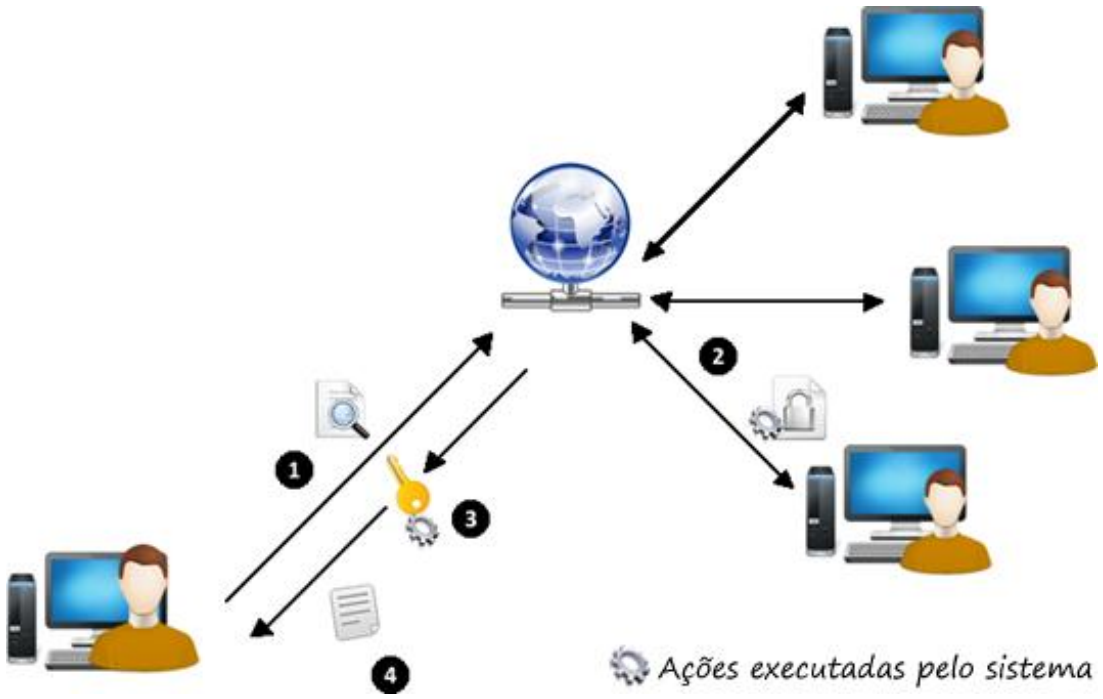


Figura 28 - Recuperar um arquivo

Para realizar o *download* de um arquivo, o usuário precisa recuperar a lista que contém os arquivos disponíveis. Na lista o usuário pode escolher o arquivo desejado para recuperar, na Figura 28 são mostrados os passos executados ao realizar um *download:*

- Passo 1 - é realizada uma busca para verificar se o arquivo continua disponível;
- Passo 2 – o arquivo solicitado é enviado ao par que fez a requisição. O arquivo enviado chega ao par criptografado;
- Passo 3 – processo de decifrar o arquivo e a chave utilizada é a senha que o usuário configurou na aplicação. Feito o processo de decifrar o arquivo, então é exibido para o par que solicitou.

Os cenários de uso apresentados foram baseados nas possíveis ações descritas nos casos de uso, e foram utilizados como base para o desenvolvimento

da aplicação e o roteiro de execução dos códigos de implementação, que são detalhados na próxima seção.

4.5 Implementação da aplicação *Peer Backup*

Nesta seção são apresentadas partes relevantes do código e as telas da aplicação desenvolvida. Durante o desenvolvimento da aplicação foi determinada uma sequência de ações, que é demostrada na seguinte ordem:

- Inicialização da aplicação:
  - Verificação de dados (banco de dados e pastas);
  - Criação e autenticação da rede;
  - Conexão do usuário com o sistema;
- Interface gráfica:
  - Acesso à tela inicial;
  - Envio de arquivos;
  - Recuperação de arquivos
  - Verificação de pares conectados;
  - Acesso aos Logs de eventos.

A sequência descrita representa o comportamento do sistema desde o início da execução até a apresentação da parte gráfica para o usuário, e são apresentadas nas seções seguintes.

Iniciando aplicação

A fase de inicialização da aplicação funciona como um modelo de dependências, na qual uma ação só pode ser executada se a anterior tiver sido iniciada ou completada. A Figura 29 mostra a sequência de ações referentes à inicialização da aplicação.

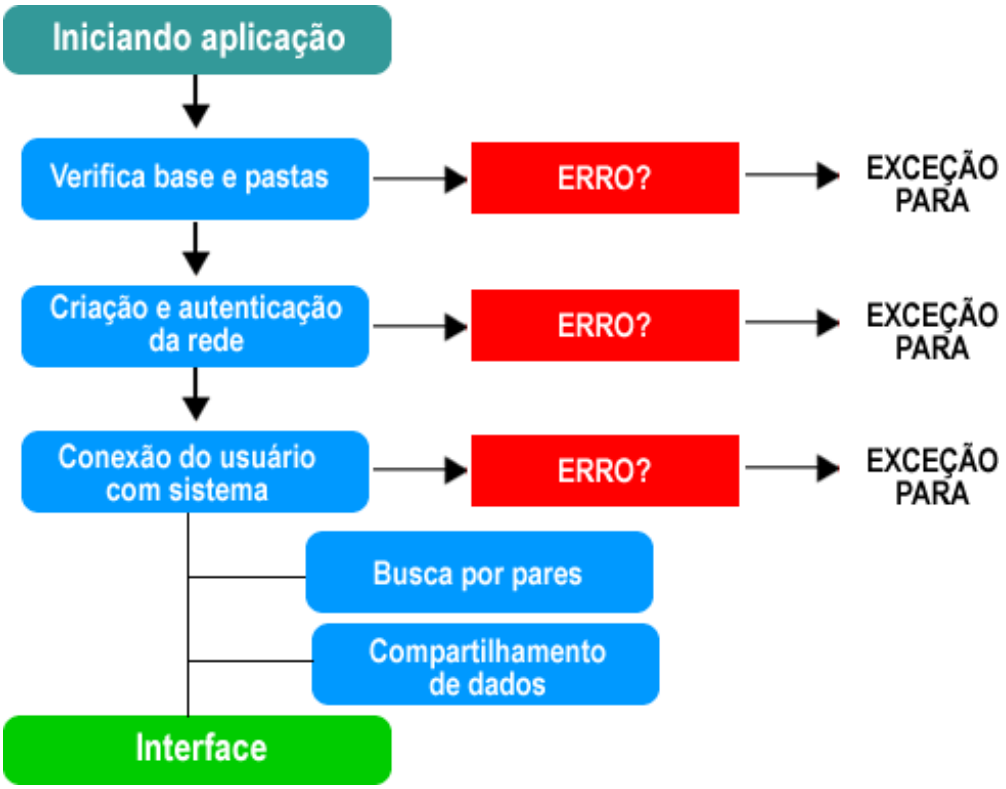


Figura 29 - Sequência de eventos na inicialização do *Peer Backup*

Na Figura 29 é mostrada a sequência de ações que o sistema executa para mostrar a interface ao usuário, no qual “Iniciando aplicação” representa uma ação do usuário, os quadrados de cor azul representa ações do sistema, os objetos de cor vermelha representam condicionais no sistema, as exceções são erros ocorridos durante os processos e finalizam a aplicação, por fim a cor verde, que representa telas do sistema.

Na inicialização do sistema qualquer erro gerado por algum dos métodos aborta a aplicação, isto porque as próximas requisições precisam do resultado positivo gerado na anterior, como mostrado na Figura 29, nos itens “verifica base e pastas”, “criação e autenticação da rede” e “conexão do usuário com o sistema”. A primeira etapa da inicialização é verificar a base de dados e as pastas necessárias para o funcionamento da aplicação, pois informações sobre o usuário como *login*, chave de segurança, arquivos enviados estão presentes nesses arquivos e são utilizados nos próximos métodos, além das pastas de arquivos recuperados e a pasta em que os outros pares guardam suas cópias.

Todos os comandos P2P na rede são enviados para os serviços iniciados na etapa de criação e autenticação da rede. Nesta etapa, os pares são anunciados para

os demais, fornecendo seu endereço JXTA. Erros de localização da rede e de autenticação podem acontecer.

A conexão do usuário com o sistema é a última etapa em que pode ocorrer um erro que gere uma exceção e pare o sistema. Esta fase consiste, basicamente, na conexão do usuário ao sistema, utilizando sua chave secreta. Um erro comum nessa etapa é digitação de chave inválida ou errada pelo usuário, o que impossibilita que a aplicação continue sua execução.

Verificação de dados

A verificação da base de dados e das pastas do sistema são necessárias, pois as pastas “Config”, “Downloads” e “Share” são locais em que são armazenados os arquivos dos usuários e as configurações da aplicação. O arquivo “Base.xml” também é necessário, pois, informações sobre os arquivos enviados são gravadas nele. A Figura 30 mostra a sequência da criação dos dados.

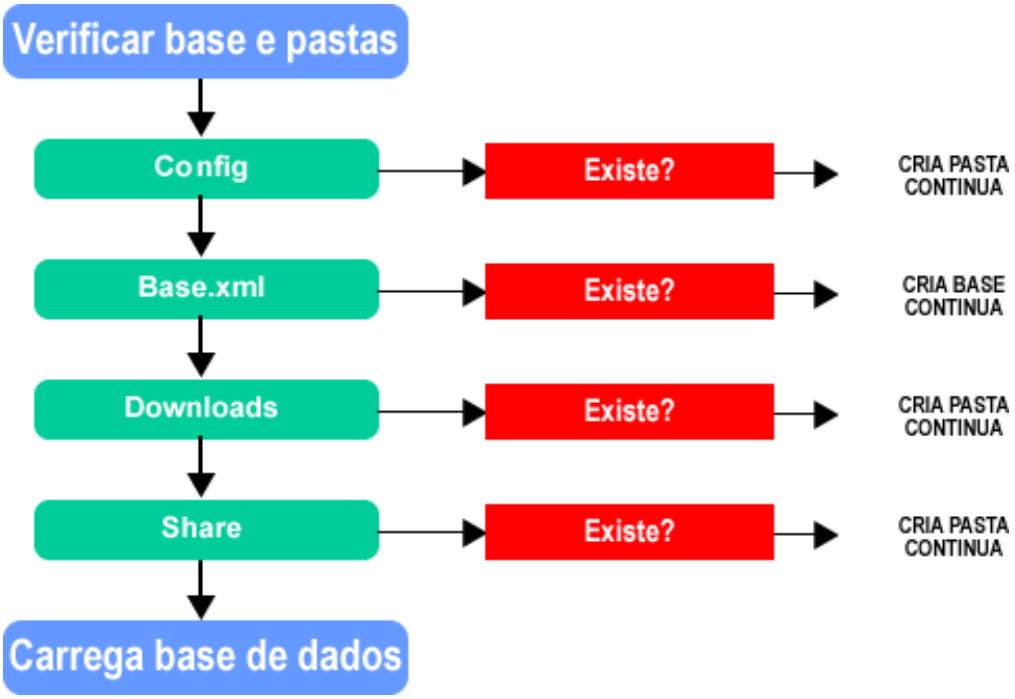


Figura 30 - Fluxo verificar base e pastas

Na Figura 30 é mostrada a sequência de criação das pastas e da base, a necessidade de cada um dos itens são detalhados na seção 4.2, no qual os objetos de cor verde claro representam as pastas e arquivos a serem criados. Antes da

criação das pastas é feita a verificação de sua existência, pois, se existir não é necessário criar. As pastas e arquivos são verificados e criados nessa ordem: “Config”, “Base.xml”, “Downloads” e “Share”. Após concluir a criação das pastas, o sistema carrega o arquivo “Base.xml” para a memória e continua a execução, que é a criação da rede.

Criação e autenticação da rede

O processo de criação da rede é executado pela aplicação, mesmo que exista alguma rede já ativa. Isto ocorre porque é preciso iniciar os serviços de busca de uma rede genérica. A Figura 31 mostra os eventos executados para inicialização da rede.

Figura 31 - Criação da rede P2P

Todo o processo de inicialização e autenticação da rede P2P, mostrado na Figura 31, é feito de forma transparente ao usuário. Na qual é iniciada uma rede, contendo os serviços de uma rede P2P. Os serviços de busca são utilizados para encontrar um serviço de rede existente. Como é mostrado na Figura 31, existem

duas possibilidades quando se verifica a existência de uma rede P2P, a rede estar ativa ou a rede não existir.

Quando a rede é encontrada, o sistema cria uma cópia do objeto de referência da rede, que inclui localização, nome de grupos, pares ingressos e anúncios transmitidos. Caso o par servidor se desconecte, o objeto copiado é utilizado para iniciar uma nova rede contendo os pares existentes. Por fim o sistema autentica o usuário na rede e realiza o procedimento para incluir o par no grupo e anunciá-lo para os outros pares.

Ao buscar e não encontrar um serviço de rede disponível, o próprio par é eleito como servidor. Para isso, é necessário somente autenticar o usuário na rede, proceder com o ingresso do par no grupo criado e aguardar que novos pares se conectem.

Para encontrar redes disponíveis a aplicação utiliza o nome definido para a rede P2P. Para a aplicação *Peer Backup*, foi definido o nome “PeerBackup” para qualquer rede criada a partir dela, sendo que sistema busca por esta referência. A Listagem 2 mostra o trecho de código que implementa as buscas por redes P2P.

```
78  public boolean Pesquisar_Grupo_Existente() {
        Enumeration publica = null;
80      Log.append("\nBuscando serviços de grupo existentes");
81      for (int i = 0; i < 5; i++) {
82          try {
83              publica = this.DS_Padrao.getLocalAdvertisements(DiscoveryService.GROUP, "Name",
84                      "PeerBackup");
85              if ((publica != null) && publica.hasMoreElements()) {
86                  this.PGA_Backup = (PeerGroupAdvertisement) publica.nextElement();
87                  this.Grupo_Backup = this.GrupoGlobal.newGroup(PGA_Backup);
88                  Ingressar_Grupo(this.Grupo_Backup);
89                  return true;
90              } else {
91                  this.DS_Padrao.getRemoteAdvertisements(null, DiscoveryService.GROUP, "Name",
92                          "PeerBackup", 1);
                    Thread.sleep(10000);
94              }
95
97          } catch (Exception e) {
                System.out.println(e.getMessage());
98          }
99      }
100     return false;
101 }
```

Listagem 2 - Método de busca por rede P2P

No método `Pesquisar_Grupo_Existente()` da classe `JXTA_Service`, que é mostrado na Listagem 2, é realizada a busca por redes P2P que possuam o

atributo “Name” igual a “PeerBackup”. Este processo é executado cinco vezes e aguarda dez segundos para cada nova busca.

- linha 79 - é criada uma variável para armazenar uma coleção de objetos;
- linha 80 - adicionadas informações ao log de eventos;
- linha 83 - é utilizado o método `getLocalAdvertisements()` da classe `DiscoveryService` do JXTA, que retorna todos os elementos referentes a grupos encontrados na busca na rede. Os argumentos informados ao método são, respectivamente, o tipo da busca (no caso, grupo), o atributo a ser pesquisado e o que deve conter no atributo. Esse método realiza a busca utilizando os endereços JXTA contido nos metadados guardados na máquina do usuário referente à anúncios de redes encontradas em outro momento;
- Na linha 85 - é verificada a existência de algum elemento, quando encontrado são executadas as linhas 86 e 87;
- linha 86 - realiza uma cópia do objeto que é responsável pelos serviços de anúncios JXTA;
- linha 87 - é efetuada uma cópia do objeto da rede;
- linha 88 - inclui o par na rede;
- linha 91 - caso não encontre nenhuma rede P2P utilizando o cache, então é feita a busca por inundação, enviando anúncios para todos os pares disponíveis na rede. O método utilizado para propagar a requisição é o `getRemoteAdvertisements()`, que recebe como atributos respectivamente o par especifico a buscar, no caso *null*, pois é para todos, tipo de anúncio, atributo a ser buscado, o valor do atributo, e a quantidade de respostas esperadas; e
- linha 93 – a aplicação executa esse processo de busca em cache e por inundação, cinco vezes com um intervalo de 10 segundos;

Como mostrado na Figura 31 uma rede é criada antes mesmo da busca por outras existente, sendo utilizado apenas os serviços de localização. Quando não encontra uma rede ativa, automaticamente é criado um grupo P2P, no qual o usuário é conectado. Esse grupo é anunciado como uma rede P2P para que novos pares consigam conectar. O código utilizado para criar o grupo é mostrado na Listagem 3.

```
150  private PeerGroup Criar_Novo_Grupo() {
151      PeerGroup novo = null;
152      Log.append("\nCriando serviços de grupo.");
153      try {
154          ModuleImplAdvertisement Anuncio = GrupoGlobal.getAllPurposePeerGroupImplAdvertisement();
155          PeerGroupID Grupo_Id = (PeerGroupID) IDFactory.fromURL(new URL("urn", "",
156                  Grupo_Backup_ID));
157          novo = GrupoGlobal.newGroup(Grupo_Id, Anuncio, "PeerBackup",
158                  "PeerBackup rede de compartilhamento");
159          PGA_Backup = novo.getPeerGroupAdvertisement();
160          DS_Padrao.publish(PGA_Backup);
161          DS_Padrao.remotePublish(PGA_Backup);
162
163      } catch (Exception ex) {
164          Logger.getLogger(JXTA_Service.class.getName()).log(Level.SEVERE, null, ex);
165      }
166      return novo;
167  }
```

Listagem 3 - Código criar novo grupo

O código mostrado na Listagem 3 é executado quando a aplicação cria um novo grupo, isto ocorre quando não é encontrado um grupo ativo. O código executa conforme a seguinte sequência:

- linha 154 - é criado um objeto de `ModuleImplAdvertisement`, uma classe do JXTA responsável pelos serviços de anúncios. A variável `Anuncio` recebe como valor os serviços de anúncios referentes à grupos, isso por meio do método `getAllPurposePeerGroupImplAdvertisement`();
- Linha 154 - variável `GrupoGlobal` é referente ao grupo criado no momento que a aplicação foi iniciada, e foi utilizada apenas para descobrir redes disponíveis e implementar os serviços P2P;
- linha 155 - é criada uma variável para identificação do grupo, no qual é gerado um endereço URN (Nome uniforme de recurso), que é o identificados utilizado para localizar o grupo na rede;
- linha 157 - criado um novo grupo criado que recebe como parâmetros no construtor do método a identificação do grupo, os serviços disponíveis de grupo, o nome do grupo e a descrição. Após a criação do grupo é criado um anúncio para publicá-lo na rede;
- linha 159 – criado um anúncio para publicar o par, na qual a variável `PGA_Backup` que é do tipo `PeerGroupAdvertisement`, classe responsável por criar anúncios de grupo. A variável recebe o anuncio referente ao grupo que foi criado;
- linha 160 e 161 – par publicado em cache e na rede utilizando o método `publish` da classe `DiscoveryService`.

O grupo criado fica disponível na rede para novos pares se conectarem e enviarem seus arquivos para backup.

Conexão do usuário com o sistema

Antes de obter acesso a interface gráfica o usuário precisa cadastrar uma senha para criptografia dos arquivos ou fazer *login* caso já tenha cadastrado. A Figura 32 mostra o processo executado para o usuário obter acesso à interface da aplicação.

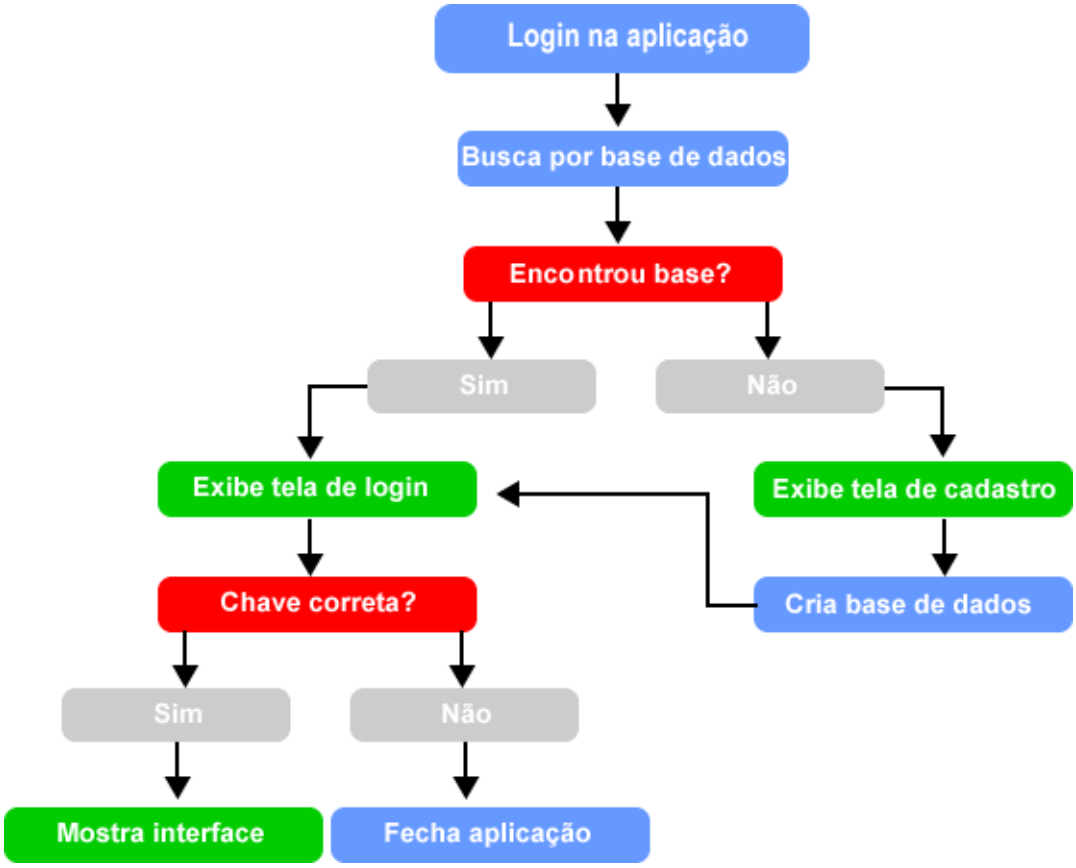


Figura 32 - Etapas de acesso à aplicação

As etapas do processo de *login* mostrado na Figura 32 são necessárias para que o usuário tenha acesso à interface gráfica da aplicação. Na primeira etapa é verificada a existência da base de dados com as informações do usuário. Essa base consiste na pasta “.jxta” e no arquivo “Profile.ini” que é criado na pasta “Config”.

Interface do sistema

A interface é responsável por receber interações do usuário para realizar ações na aplicação. Sendo que as informações exibidas ao usuário são buscadas na base de

dados existente, que são dados referentes a nome do usuário, arquivos etc. Quando não encontra a base o sistema exibe primeiramente a tela do JXTA para cadastro de usuário e senha, que é mostrada na Figura 33.

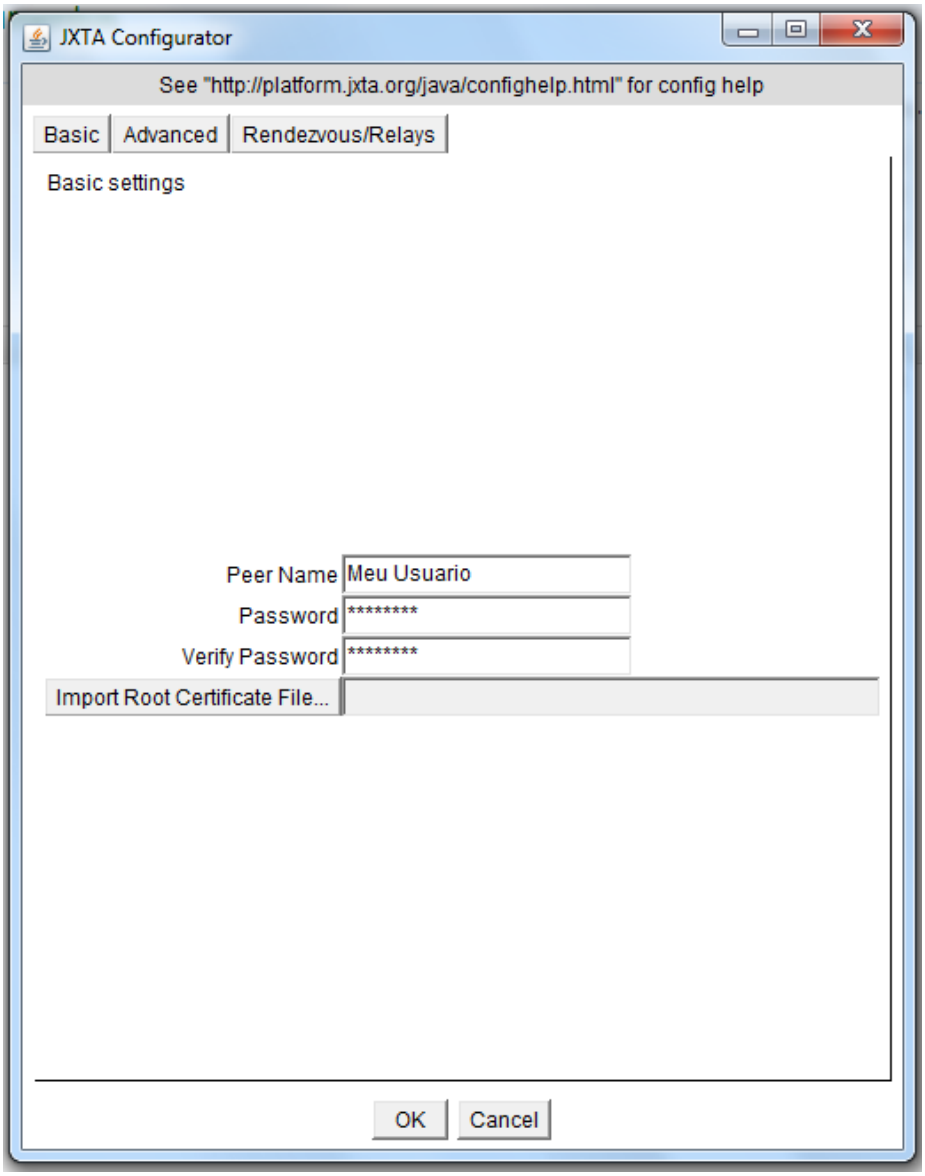


Figura 33 - JXTA Configurator

A Figura 33 mostra a tela referente ao "JXTA *Configurator*", que é uma interface de configuração fornecida pelo JXTA, no qual é aberto automaticamente quando o usuário não realizou a configuração da tecnologia. O campo *Peer Name* é o nome que o usuário deseja utilizar na rede, sendo considerado como seu *login* e deve ser utilizado sempre o mesmo, pois quando é feita a recuperação de arquivos, o sistema usa como referência o nome do usuário. O campo *Password* deve ser

preenchido por uma senha do usuário de 8 ou mais caracteres, e o campo *Verify Password* é apenas a confirmação da senha.

Os botões mostrados no topo da Figura 33 são configurações adicionais do JXTA, que deve ser modificada por um usuário com experiência na tecnologia ou pelo desenvolvedor da aplicação, caso seja necessária a utilização de outras configurações diferentes da padrão. As configurações que podem ser alteradas são:

- Log;
- Proxy;
- Protocolo TCP;
- Protocolo HTTP.

Após realizar a configuração utilizando o JXTA *Configurator*, automaticamente é aberta uma janela para a configuração da senha de criptografia que pode ser igual a digitada na configuração do JXTA. A chave de criptografia é utilizada para conectar-se no sistema e cifrar/decifrar os arquivos do usuário. Após digitar os dados é criada a pasta “.jxta” com os metadados do JXTA e o arquivo “Profile.ini” que possui o nome do usuário digitado no JXTA *Configurator* criptografado com a chave de criptografia.

Após feita a criação da base é mostrada a tela de *login*, que o usuário deve digitar a chave de criptografia para acessar o sistema. Para verificar a autenticidade da chave, é feita a decifragem do arquivo “Profile.ini”. A Figura 34 mostra o processo de verificação da chave.

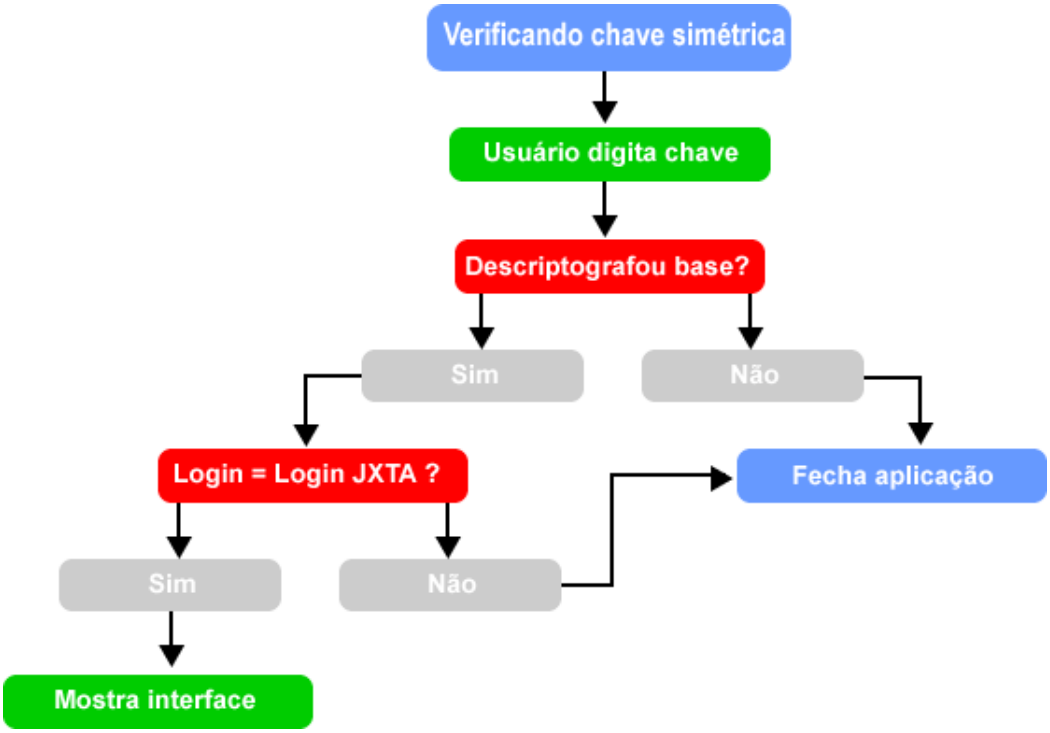


Figura 34 - Verificação de chave

A verificação feita no login é utilizando a chave simétrica como mostra a Figura 34. Quando o usuário digita a chave, é feita a tentativa de decifragem do arquivo “Profile.ini”, que contém criptografado o nome de *login* definido no JXTA *Configurator.* Caso seja possível a descriptografia utilizando a chave fornecida pelo usuário, então é feita a verificação apenas para conferir se a base existente do JXTA é referente a base de dados da aplicação. Então é verificado o nome de usuário JXTA e o da base, caso seja igual é mostrada a interface gráfica. Quando a chave é inválida ou os dados encontrados na base da aplicação são diferente do JXTA então a aplicação é encerrada. Um usuário pode se passar por outro, pois a verificação é apenas para verificação local, mas não conseguirá recuperar os arquivos sem a chave de descriptografia.

Após esta etapa, então, é exibida ao usuário a tela da aplicação, que possui as funcionalidades de *backup* como envio, busca e recuperação de arquivos. Além da tela de progresso de download, lista de arquivos enviados, logs etc. A tela inicial da aplicação mostra a lista dos arquivos enviados pelo usuário, caso não tenha enviado nenhum, a lista fica em branco, como mostrado na Figura 35.

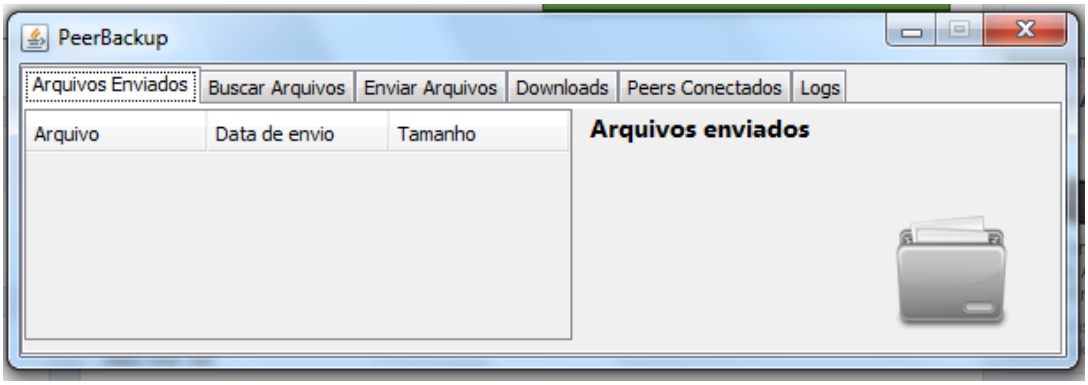


Figura 35 - Interface gráfica *Peer Backup*

As telas da aplicação são acessadas por meio das abas disponíveis na interface, como mostrado na Figura 35. A aba “Buscar Arquivos” disponibiliza uma lista de arquivos disponíveis que podem ser recuperados. Para fazer o envio de arquivos o usuário deve acessar “Enviar Arquivos. Em “Downloads” é mostrado o status dos arquivos que estão sendo recuperados. Em “Peers Conectados” o usuário da aplicação pode verificar quais pares estão disponíveis e conectados na rede. As requisições feitas pelo sistema e erros são mostrados em “Logs”.

Envio de arquivos

Os arquivos do usuário são enviados para os outros pares por meio da aba “Enviar Arquivos” e são criptografados com a chave do usuário utilizando a tecnologia de criptografia AES, antes de serem compartilhados na rede. A cifragem do arquivo é a forma de garantir a confidencialidade da informação, na Listagem 4 mostra o código que cifra os arquivos.

```
198             OutputStream Arquivo_Criptografado = new FileOutputStream(Arquivo);
199
200             byte[] Chave_Bytes = chave.getBytes();
201             SecretKeySpec Chave_Secreta = new SecretKeySpec(Chave_Bytes, "AES");
202             Cipher Cifrador = Cipher.getInstance("AES");
203
204             Cifrador.init(Cipher.ENCRYPT_MODE, Chave_Secreta);
205             CipherInputStream CIS = new CipherInputStream(Arquivo_Original, Cifrador);
206
207             //Copiando para pasta Share
208             byte[] bytes = new byte[64];
209             int numBytes;
210             while ((numBytes = CIS.read(bytes)) != -1) {
211                 Arquivo_Criptografado.write(bytes, 0, numBytes);
212             }
213
214             Arquivo_Criptografado.close();
            CIS.close();
```

Listagem 4 - Código de cifragem AES

O trecho de código mostrado na Listagem 4 é uma parte do método `Criptografar()` da classe `Arquivo`, que é responsável por todas as manipulações de arquivo como criptografia, cálculo de *hash*, etc. O código segue uma sequência de execução:

- linha 197 - criada uma instância `OutputStream`, objeto utilizado para auxiliar na escrita dos *bytes*, que recebe como parâmetro o arquivo que será criptografado.
- linha 199 - a chave do usuário é transformada em *bytes*, para ser usada na construção da chave secreta de criptografia do tipo AES, na linha 200.
- linha 201 – a cifragem do arquivo é feita utilizando a biblioteca `crypto` nativa do Java, a classe `Cipher` faz parte dela, recebendo como parâmetro o tipo de criptografia, no caso AES.
- linha 203 - o método init da classe `Cipher` inicia o cifrador e configura o modo de operação, recebendo como parâmetro a operação, no caso criptografar, e a chave transformada em *bytes*.
- linha 204 - a variável `CIS`, do tipo `CipherInputStream`, é criada para ler os bytes antes que seja criptografado, para garantir que o conteúdo pode ser lido antes de ser escrito.
- linhas 209 a 211 é feita a leitura dos dados a cada 64 bytes, que depois são escrito em arquivo físico e colocado na pasta Share.

Além da cifragem do conteúdo dos arquivos, também é feita a codificação no nome dos arquivos antes de enviá-los para a rede. O nome do arquivo segue um prefixo estabelecido na aplicação antes de ser codificado, que é:

- NOMEDOUSUARIO__DIA-MÊS-ANO-HORAMINUTO__NOMEDOARQUIVO;
- Exemplo: TARIK__13-06-2013-1510__FOTOS.ZIP.

Após o nome ser modificado de acordo com o prefixo, o mesmo é codificado utilizando o algoritmo base64, cuja codificação é reversível. Dessa forma a aplicação faz uma busca utilizando apenas o nome do usuário, quando é preciso recuperar os arquivos enviados.

Recuperação de arquivos

Quando um usuário da aplicação deseja recuperar um arquivo é necessário utilizar a aba “Buscar Arquivo”. No momento que for feita a requisição, a aplicação faz uma requisição à classe `Arquivo_Buscar_Thread`, que é o serviço responsável por iniciar uma instância da classe `ListRequestor`, como mostrado na Listagem 5.

```
42      @Override
43      public void run() {
44
45          if (opcao == 0) {
46              Log.append("\nBuscando meus arquivos...");
47              Tempo_Inicio_Thread = System.currentTimeMillis();
48              boolean loop = true;
49
50              this.JTable.setEnabled(false);
51              while (loop) {
52                  List_Requestor = new ListRequestor(Grupo_Backup, Query_Busca, JTable, 0, arquivos);
53                  List_Requestor.activateRequest();
54
55                  if ((System.currentTimeMillis() - Tempo_Inicio_Thread) > 30000) {
56                      Log.append("\nBusca por arquivos finalizada.");
57                      Botao.setEnabled(true);
58                      Carregando.setVisible(false);
59                      loop = false;
60                  }
61              }
62              this.JTable.setEnabled(true);
63          }
64          else if(opcao == 1)
65          {
66              boolean loop = true;
67              while(loop){
68                  List_Requestor = new ListRequestor(Grupo_Backup, "*", null, 1, null);
69                  List_Requestor.activateRequest();
70              }
71          }
```

Listagem 5 - Método run() da thread Arquivo_Buscar_Thread

Quando a thread Arquivo_Buscar_Thread, mostrado na Listagem 5 é iniciada, uma verificação é realizada com intuito de distinguir as requisições, isto é, verificar se a requisição foi feita pelo usuário, no caso busca por arquivos, ou pela aplicação, arquivos de outros pares compartilhados na rede como mostrado na sequência:

- linha 44 - iniciada a verificação da requisição, caso seja uma requisição do usuário (opcao == 0), então a condição é verdadeira e a aplicação executa as linhas seguintes 45-61. A variável opção é importante para a aplicação, pois define se a consulta deve retornar um resultado para o usuário (lista de arquivos encontrados) ou se é uma requisição do sistema com objetivo de efetuar *download* de arquivos compartilhados na rede;

- linha 46 - é gravado na variável `Tempo_Inicio_Thread` o valor da hora no momento que o sistema executou a linha, para controlar o tempo de execução da *thread*. Para recuperar um arquivo é utilizada a classe `ListRequestor`;
- linha 51 - Na variável `List_Requestor`, que é passado como parâmetro o grupo *peer-to-peer* ativo, o nome do arquivo (`Query_Buscar`), a tabela que exibe os resultados (`JTable`), o valor correspondente a opção para definir de quem veio a requisição (no caso do usuário, 0) e a lista de arquivos em que os resultados são salvos.

A classe `ListRequestor` herda as características da classe `CachedListContentRequest` do JXTA, que é responsável por buscas. Para que se adeque às necessidades do sistema foi modificado o método que informa os resultados obtidos pela busca, o `notifyMoreResults()`. A Figura 36 mostra o funcionamento do método.

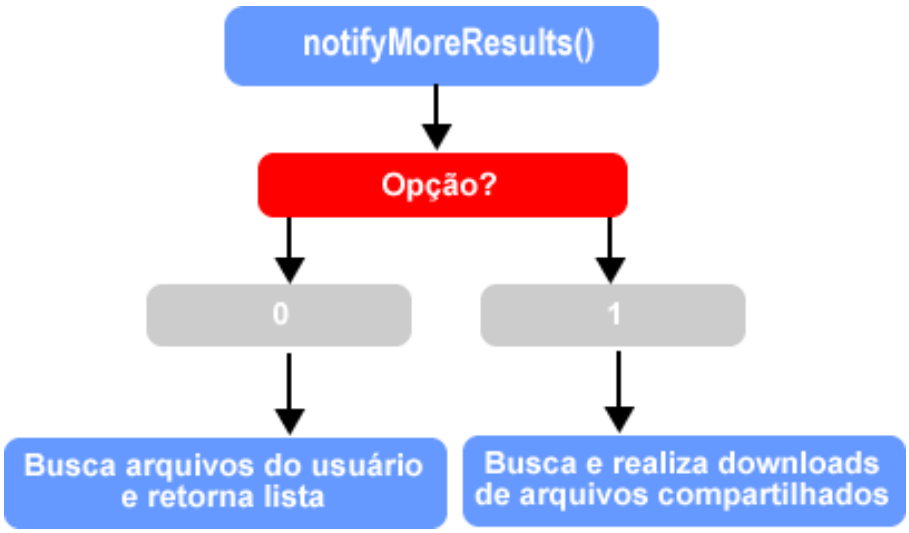


Figura 36 - Método notifyMoreResults()

O método mostrado na Figura 36 é implementado pelo JXTA e sobrescrito pela aplicação, sendo responsável por notificar resultados de buscas. Quando executado, o método verifica se a requisição está sendo feita por um usuário ou pelo sistema. Caso seja uma requisição do próprio sistema, então é feita a busca por todos os arquivos de todos os usuários, e quando encontra realiza o download automaticamente com intuito de propagar as cópias pela rede. Quando a busca é feita por usuário, então é realizada uma busca pelos arquivos enviados pelo usuário

que por fim retorna a lista de arquivos encontrados disponíveis para *download* na aba “Buscar Arquivos” da interface, como mostrado na Figura 37.

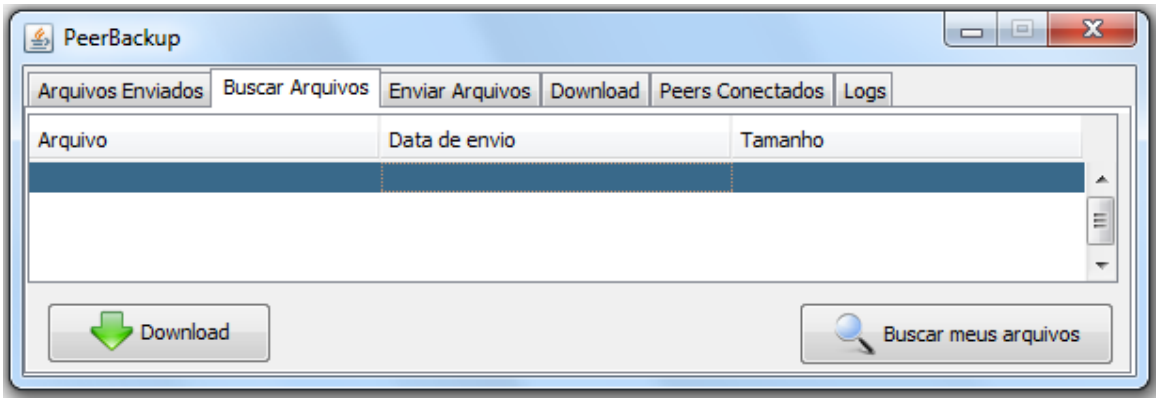


Figura 37 - Aba Buscar Arquivos

Na aba “Buscar Arquivos” mostrada na Figura 37, é o local no qual os usuários da aplicação podem recuperar seus arquivos enviados. Ao utilizar o botão “Buscar meus arquivos” o sistema busca por dois minutos anúncios de arquivos que contenham no nome o prefixo referente ao nome do usuário, após o término do tempo é exibido os resultados encontrados. Isto é, ao realizar a requisição para a classe `ListRequestor`, no atributo `Query_Buscar` é passado o valor “NOMEDOPAR*”, sendo que qualquer valor que possua o nome do par no início do arquivo é considerado dele. A query de busca é feita deste modo porque os arquivos são renomeados antes de serem compartilhados com intuito de facilitar no momento da busca.

Após o retorno da lista de arquivos, o usuário pode escolher e recuperar um arquivo, utilizando o botão “Download” da tela. A requisição de download inicia um serviço da classe `Download_Thread`, que é responsável por invocar os métodos da classe `Download`, tendo como objetivo de recuperar o arquivo.

A classe `Download` herda os métodos da classe `GetContentRequest` do JXTA, que é responsável pelo download de arquivos presentes na rede. A classe `Download` recebe como parâmetro no construtor: o grupo ativo, o anúncio de conteúdo, o local que o arquivo deve ser salvo na máquina e o objeto que deve ser manipulado para exibir informações de status do *download*. Para informar sobre a situação do *download*, existem três métodos que são sobrescritos da classe superior, que são:

- `notifyUpdate`(): exibe informações de progresso em tempo real do arquivo que está sendo recuperado;
- `notifyDone`(): informa quando o download foi concluído;
- `notifyFailure`(): utilizado para informar falha no download de conteúdo.

Os métodos descritos trabalham diretamente com a aba “Download” da interface da aplicação, para informar o usuário do progresso do download. O usuário também pode cancelar um download em progresso, na Figura 38 mostra a aba “Downloads”.

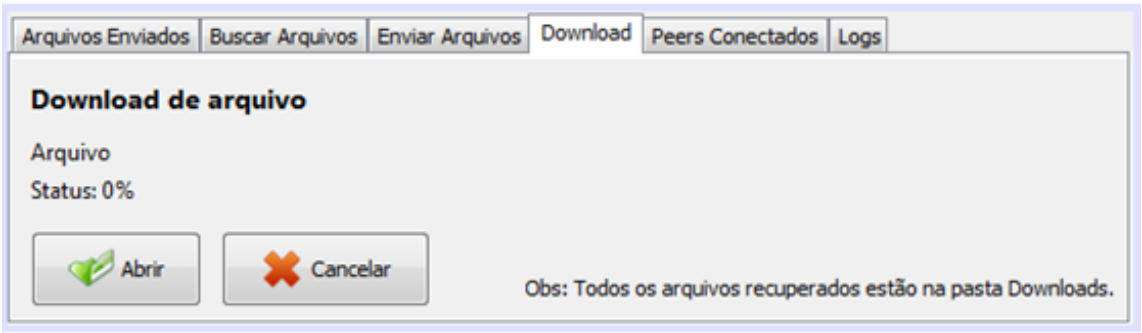


Figura 38 - Aba "Download"

Ao realizar uma requisição de download, automaticamente o progresso é exibido na aba “Download”, como mostrado na Figura 38. Por meio da interface, o usuário pode cancelar uma requisição em andamento ou abrir o arquivo quanto o processo estiver concluído. No momento que é concluído o processo de recuperação é feita a decifragem do arquivo utilizando a chave do usuário, caso seja diferente o sistema retorna uma mensagem de erro e o arquivo não pode ser aberto.

Verificação de pares conectados

Por meio da interface gráfica o usuário pode verificar se existem pares conectados à rede, e ainda quais são eles utilizando a aba “Peers Conectados”. O JXTA fornece os métodos necessários para buscar por anúncios de pares e verificar se ainda são válidos. Este procedimento é feito pela classe `Peer_Thread`, que herda e sobrescreve o método de descoberta da classe `DiscoveryListener`, como mostrado na Listagem 6.

```
55      public void discoveryEvent(DiscoveryEvent de) {
56          DiscoveryResponseMsg Response = de.getResponse();
57          Enumeration Enum = Response.getAdvertisements();
58          Lista_Pares.setListData(Gerar_Lista_Pares(Enum));
        }
```

Listagem 6 - Método discoveryEvent classe Peer_Thread

O método `discoveryEvent`, mostrado na Listagem 6, é responsável por fazer a descoberta de pares ativos na rede e retornar os resultados. A execução do código segue desta forma:

- linha 55 - inicializada a variável `Response`, que é uma instância de `DiscoveryResponseMsg`, responsável por obter respostas de anúncios de pares na rede. A resposta enviada pelos pares são *status*, localização na rede, etc.;
- linha 56 - as respostas são numa variável do tipo `Enumeration`, que ordena o conjunto de itens e remove dados repetidos;
- linha 57 é mostrada na interface do usuário os pares disponíveis.

Na interface do usuário é mostrada a lista dos pares encontrados pela aplicação, como mostrado na Figura 39.

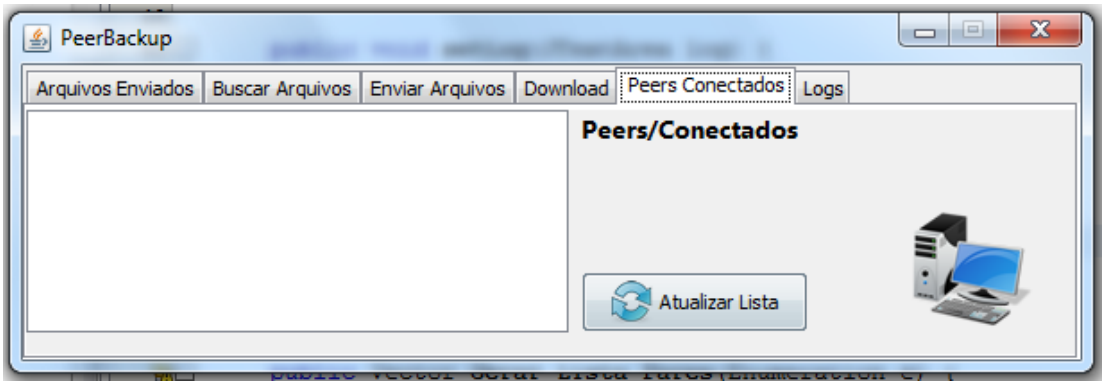


Figura 39 - Aba Peers Conectados

Quando um usuário faz uma requisição para atualizar a lista de pares conectados, utilizando o botão “Atualizar Lista” mostrado na Figura 39, a aplicação executa uma instância de `Peer_Thread` que faz a buscar de pares por dois minutos e atualiza a lista caso tenha pares conectados. Caso ocorra algum erro durante o processo, é adicionado ao log do sistema.

Acesso ao log de eventos

Os logs de eventos da aplicação serve para informar ao usuário erros ocorridos e informativos sobre serviços executados. A lista de ocorrências fica disponível para o usuário na aba Logs da aplicação, como mostrado na Figura 40.

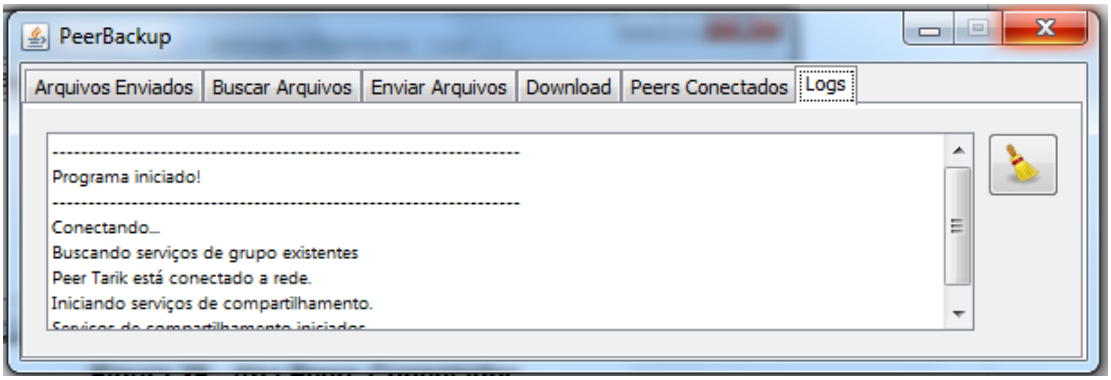


Figura 40 - Aba Logs

O registro de eventos mostrado na Figura 40, é uma forma de deixar o usuário informado de problemas ocorridos no sistema, requisições feitas por ele, etc.. Caso o usuário deseje, é possível limpar os logs mostrados.

As funcionalidades desenvolvidas podem apresentar exceções não previstas, e que podem impedir seu funcionamento correto, desta forma é preciso validar sua utilização. Desta forma foram efetuados testes de interface, rede, funcionalidades e também de autenticidade, e são detalhados na próxima seção.

4.6 Testes realizados sobre a aplicação

Segundo Tomelin (2000, p.4), “atividade de teste é o processo de executar um programa com a intenção de descobrir um erro”. Além de descobrir erros, testes de *software* são realizados para a validação das funcionalidades de uma aplicação, isto é, verificar se os requisitos propostos são atendidos.

Para a realização dos testes, foi utilizado um ambiente virtual com três máquinas, com o sistema operacional Windows XP, criado pelo aplicativo VMWare. As configurações de hardware são indiferentes, necessitando apenas a instalação do Java versão 7.

Para verificar o funcionamento da aplicação *Peer Backup*, foram criados três usuários fictícios. Além disso, foi seguido um roteiro para cada teste, que consiste em:

- Testes de interface e métodos: verificar as funcionalidades propostas no projeto.
    - Inicialização da aplicação e rede;
    - Envio de arquivo;
    - Busca por pares;
    - Busca e recuperação de arquivos.
- Teste de confidencialidade dos arquivos enviados: verificar se os arquivos enviados podem ser lidos somente pelo dono que possui a chave de criptografia.

Os testes foram feitos de forma manual, simulando as ações que seriam realizadas por usuários comuns.

Teste de Interface

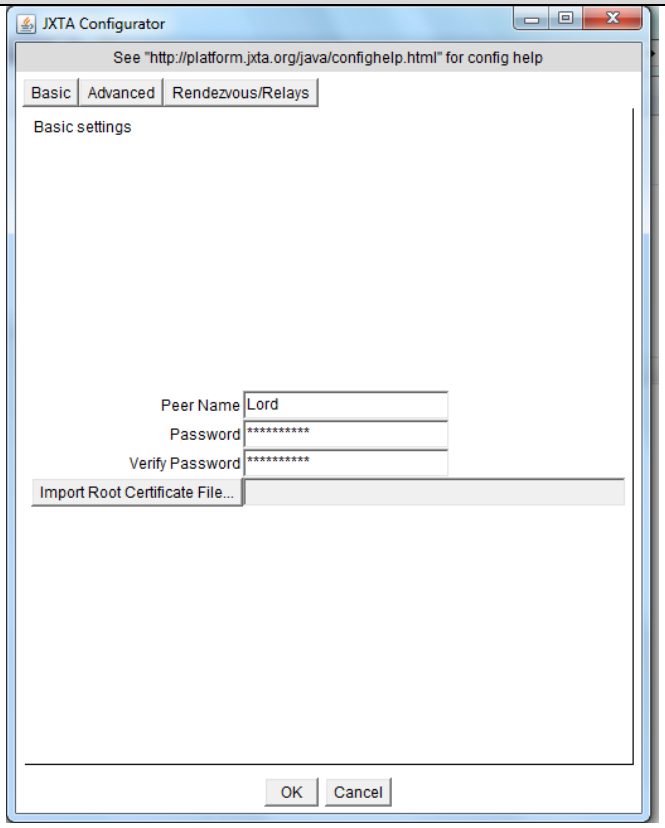
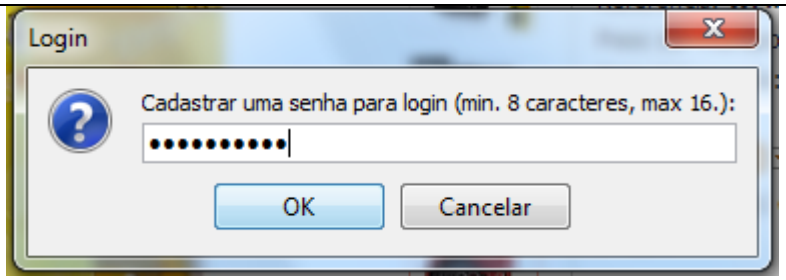
O teste de interface foi iniciado com a execução da aplicação, e foi utilizado um usuário denominado “Lord” para simular um usuário real. Para o usuário foram definidos seu login, senha e os arquivos a serem enviados, como mostrado na Tabela 3.

Tabela 3 - Atributos par Lord

Login/Nome do Par	Lord
Senha/Chave de criptografia	1234567890
Arquivos	1 arquivo de imagem tipo JPG 1 arquivo compactado tipo ZIP 1 arquivo de texto tipo TXT

Os dados *login* e senha, exibidos na Tabela 3, foram utilizados na configuração da aplicação, que é feita no JXTA *Configurator* e na configuração da chave de criptografia utilizada para o usuário Lord, como mostrado na Tabela 4.

Tabela 4 - Definindo usuário e senha da aplicação

Ação	Captura de tela
Definindo “Peer Name” e “Password” do JXTA.	
Definindo chave de criptografia e *login*.	
Interface da aplicação após configuração.	

Após a execução e a definição de *login* e senha da aplicação do usuário, como mostrado na Tabela 3, automaticamente foram criados os arquivos e pastas necessários para o funcionamento da aplicação. As pastas criadas foram: “.jxta”, “Config”, “Share” e “Downloads”. Já os arquivos criados foram: metadados do JXTA, “Base.xml” e “Profile.ini”. A interface gráfica do sistema é mostrada ao usuário

somente quando a rede P2P está ativa e quando os arquivos de dados estão criados, confirmando então a funcionalidade da rede.

Para ser possível criar a rede e realizar os próximos testes, foi necessário criar mais dois usuários fictícios para simular um ambiente de rede real. Para cada usuário foi criada uma máquina virtual e foram definidos usuários e senhas, mostrados na Tabela 5.

Tabela 5 - Usuários de teste

Login/Nome do Par	Senha/Chave de criptografia
Linus	abc112233
Raider	02201001

Os usuários mostrados na Tabela 5 foram criados para que sejam feitos os testes de compartilhamento de arquivos, que inclui envio, busca e recuperação da informação.

Envio de arquivo

O usuário “Lord” enviou os três arquivos mostrados na Tabela 6 e obteve sucesso. Sendo que o tamanho dos arquivos não foram alterados pela criptografia.

Tabela 6 - Arquivos enviados pelo usuário Lord

Arquivo	Resposta do sistema	Nome do arquivo gerado
Desktop.zip	Compartilhado com sucesso!	TG9yZA==#3#2__MTUtNS0yMDEzLTgzM19EZXNrdG9wLnppcA==#3#2
IMG_20121208_182054.jpg	Compartilhado com sucesso!	TG9yZA==#3#2__MTUtNS0yMDEzLTgzNF9JTUdfMjAxMjEyMDhfMTgyMDU0LmpwZw==#3#2
concurso.txt	Compartilhado com sucesso!	TG9yZA==#3#2__MTUtNS0yMDEzLTgzMl9jb25jdXJzby50eHQ=#

		3#2

Os arquivos mostrados na Tabela 7 foram compartilhados e a aplicação que estava sendo executada nos outros pares armazenou os arquivos para a pasta “Share” local. O sucesso no envio dos arquivos foi confirmado por meio da aba “Arquivos Enviados”, mostrado na Figura 41.

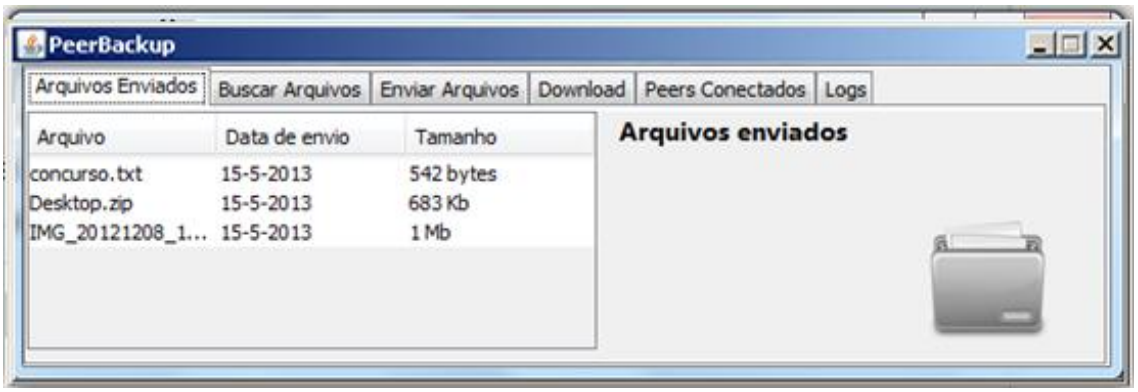


Figura 41 - Arquivos enviados por Lord

Depois de confirmar o envio dos arquivos por meio, como exibido na Figura 41, foi necessário verificar se os demais pares da rede realizaram armazenaram os arquivos do par Lord. Para confirmar foi verificada a pasta “Share” dos pares Linus e Raider. Na verificação constatou-se que a pasta “Share” dos dois nós possuía os três arquivos gerados mostrados na Tabela 6. Os arquivos são mostrados na Figura 42.

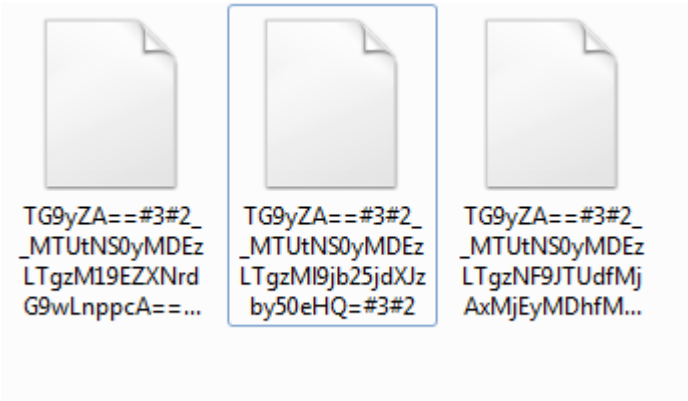


Figura 42 - Arquivos da pasta Share dos pares

Os arquivos não possuem extensão e possuem seu nome codificado como mostrado na Figura 42. Estes arquivos são replicados pelo sistema entre os pares ativos na rede.

Busca por pares

O próximo passo dos testes foi verificar se existem pares conectados na rede para que seja feita uma busca dos arquivos enviados. A verificação de pares serve para o usuário não executar ações sem necessidade, pois se não tem pares conectados, não tem arquivos disponíveis para serem recuperados.

A verificação dos pares da rede foi feita pela interface do usuário Lord, que retornou uma lista contendo os pares Linus e Raider, que é mostrado na Figura 43.

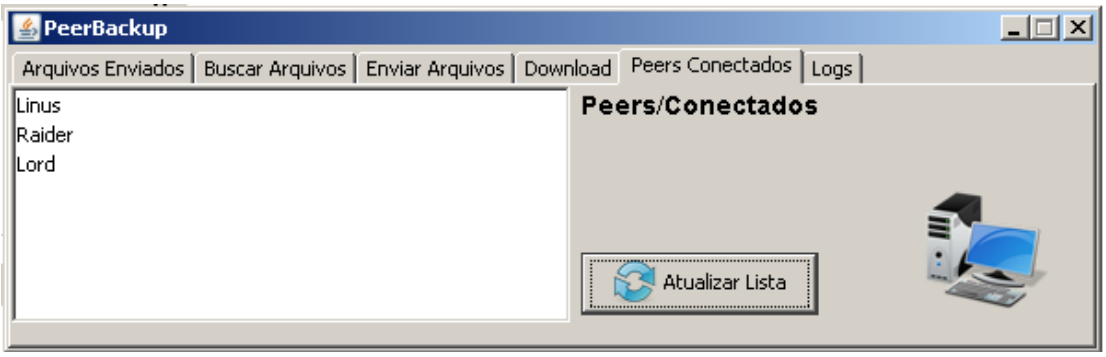


Figura 43 - Peers conectados

A existência dos pares conectados é mostrado na Figura 43, a partir disso foi feita uma busca pelos arquivos de Lord, utilizando a aba Buscar Arquivos. Ao fazer a requisição de busca, então o sistema retornou uma lista com os arquivos disponíveis, que é mostrado na Figura 44.

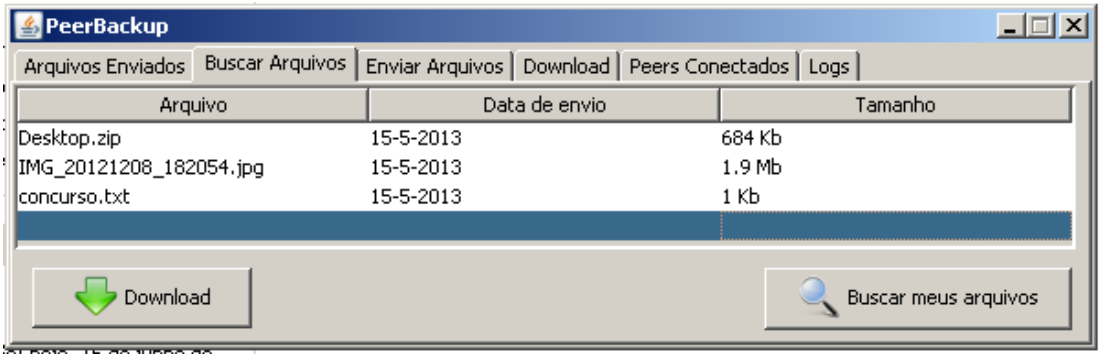


Figura 44 - Lista de arquivos disponíveis

Os arquivos mostrados na Figura 44 foram recuperados um a um, e foram guardados na pasta Downloads. Os arquivos foram descriptografados e podem ser abertos por seus respectivos *softwares*.

Esta etapa de teste realizada foi executada com o objetivo de testar as funcionalidades básicas do sistema, afim de verificar se erros poderiam afetar o seu funcionamento. Como o objetivo do trabalho é garantir que usuários não tenham acesso aos dados de outro, foi realizado o teste de confidencialidade.

Teste de confidencialidade

O teste tem o objetivo de verificar se um usuário não possa visualizar os dados enviados por outro, mesmo utilizando seu login. Como a rede não tem controle de login, então foi criado uma nova máquina virtual e realizada uma nova configuração no Peer Backup, que é mostrado na Tabela 7.

Tabela 7 - Configurações de usuário falso

Login/Nome do Par	Lord
Senha/Chave de criptografia	00000000

Os dados citados na Tabela 7 foram utilizados na configuração da chave de criptografia e do JXTA Configurator. No teste o usuário falso conseguiu visualizar a lista de arquivos enviados pelo usuário verdadeiro, mas no momento que tenta fazer download não consegue descriptografar o arquivo. Isto ocorre porque a senha de criptografia é diferente da utilizada quando o arquivo foi enviado, na Figura 45 mostra os processos de verificação.

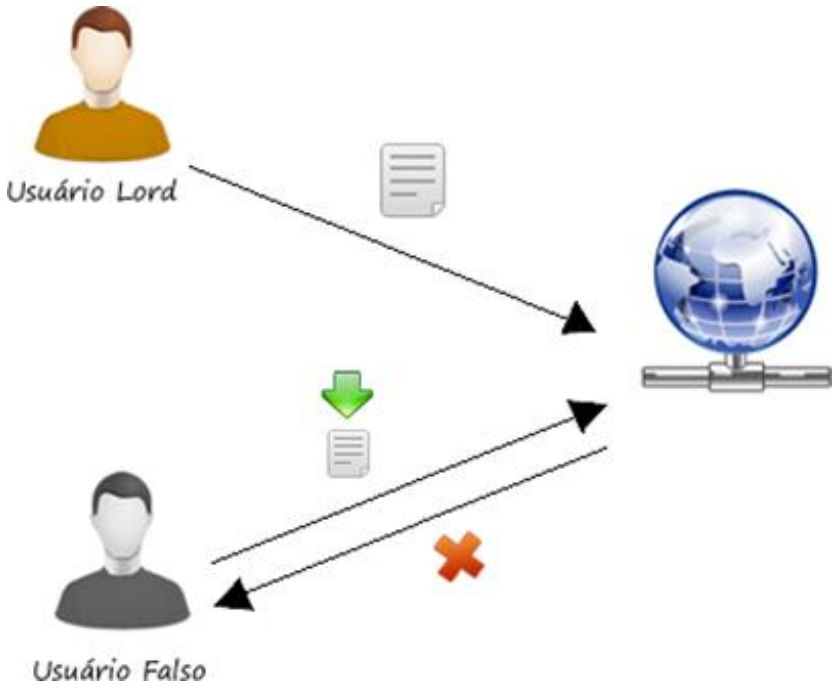


Figura 45 - Tentativa de recuperação de usuário falso

Qualquer usuário pode enviar seus arquivos quando conectado à rede e somente o dono pode recupera-lo, como mostrado na Figura 45. O usuário envia um arquivo para a rede, quando um usuário mal intencionado cria um cadastro com o mesmo *login* envia uma requisição de download mas não consegue recuperar a informação por causa da chave de criptografia.

Os testes foram realizados para verificar o funcionamento da aplicação, e também se o objetivo do projeto foi atendido.

Detalhes relevantes do projeto, dificuldades encontradas são apresentados na próxima seção.

5 CONSIDERAÇÕES FINAIS

A utilização de dados digitais tem aumentado com o passar dos anos, principalmente pela expansão da internet. Dentre estes dados estão documentos, fotos, músicas e outros objetos digitais que um usuário pode possuir no seu computador. Muitas pessoas, e até empresas, estão se desfazendo das cópias físicas e substituindo-as por cópias digitais, visando, entre outras coisas, a economia de espaço físico e a facilidade de busca.

O foco desse trabalho foi o compartilhamento de espaço em disco disponível nas máquinas de uma rede de computadores, por meio de uma arquitetura Peer-to-Peer descentralizada. Para isso, foi desenvolvida a aplicação *Peer Backup*, que possui os métodos necessários para envio, busca e recuperação de arquivos em uma rede P2P. Para acessar as funcionalidades foi implementada uma interface em que o usuário conseguisse executar as ações de *backup*.

Para realizar a manipulação da rede P2P foi utilizada a tecnologia JXTA, uma ferramenta de código aberto e disponível em várias linguagens de programação, foi um dos pontos definidos no objetivo do trabalho, pois fornece os métodos necessários para implementar a rede P2P e realizar a comunicação entre os pares da rede.

Para verificar se a aplicação oferecia as funcionalidades definidas e não apresentava problemas na utilização, foram realizados testes de interface e funcionamento, isto em um ambiente virtual, devido as dificuldades de se montar um ambiente real. Foram realizados testes de interface para verificar o funcionamento da aplicação simulando usuários de uma rede e teste de confidencialidade para garantir que somente o proprietário dos arquivos possa recuperá-lo. Nos resultados dos testes foi demostrado o funcionamento real da aplicação e as saídas obtidas durante sua execução, que ao final foi comprovado seu funcionamento.

Além dos testes básicos de funcionamento, foi realizado o de confidencialidade, pois é importante garantir que os dados não possam ser lidos lidos por outros usuários, a não ser o dono. Com os testes, foi possível concluir que

os dados armazenados em outras máquinas são legíveis apenas pelo dono, fornecendo uma segurança e confiança a mais para o usuário.

Uma das dificuldades encontradas no desenvolvimento do trabalho foi na utilização do JXTA, sendo que existe uma grande quantidade de material teórico, mas pouco pode ser encontrado quanto à implementação. A tecnologia, por sua vez, consumiu cerca de 90% da memória RAM da máquina que a está executando, o que dificultou a realização dos testes, pois havia a necessidade de executar em média 3 a 4 instâncias da aplicação ao mesmo tempo nas máquinas virtuais. O JXTA forneceu todos os métodos necessários para atender os objetivos do trabalho e possui outros recursos não explorados, durante sua utilização, que podem ser utilizados para outros tipos de aplicação P2P

Este trabalho oferece a possibilidade de continuação e aprimoramento, o que pode ser realizado em trabalhos futuros:

- Verificação de integridade dos arquivos enviados e recuperados utilizando *hash*. Umas das formas de implementar é compactar o arquivo a ser enviado junto com seu *hash* e, no momento da recuperação, realizar um novo cálculo de hash do arquivo recebido e verificar com o que está compactado;
- Utilização de um servidor apenas para autenticação dos usuários, como uma forma de reforçar a segurança na rede com intuito de impedir que usuário mal intencionados se passem por outros; e por meio desse servidor fornecer acesso da lista de arquivos enviados somente pelo dono;

A inclusão dos trabalhos propostos na aplicação podem aumentar a confiança dos usuário em questão do uso, pois garantirá a integridade dos arquivos, além da privacidade dos dados.

6 REFERÊNCIAS BIBLIOGRÁFICAS

ANDROUTSELLIS-THEOTOKIS, S., SPINELLIS, D.**A Survey of Peer-to-Peer Content Distribution Technologies**.2004.Disponível em<http://web.eecs.utk.edu/~itamar/courses/ECE-553/Project_Papers/AS04.pdf>.Acesso em 19 novembro 2012.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas Distribuídos Conceitos e Projetos**. 4. Ed. ARTMED EDITORA S.A. 2007.

FARIA, H., M. **Bacula -Ferramenta Livre de Backup**.2010.

GONG, L. **JXTA**: A Network Programming Environment. 2001. Disponível em <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=935182>. Acesso em 15 novembro 2012.

IBM. IBM **DB2 UDB versus Oracle backup andrecovery**. 2004. Disponível em <http://www.ibm.com/developerworks/data/library/techarticle/dm-0407tham/index.html>. Acesso em 15 novembro 2012.

JUNIOR, B., P. **GERENCIAMENTO CENTRALIZADO DE BACKUPS DISTRIBUÍDOS**. 2010. Disponível em <http://siaibib01.univali.br/pdf/Braz%20Pereira%20Junior.pdf>. Acesso em 19 novembro 2012.

LIPP, K., J. WHY ARCHIVE IS ARCHIVE, BACKUP IS BACKUP AND BACKUP AIN'T ARCHIVE. 1999. Disponível em <http://adsm-symposium.oucs.ox.ac.uk/1999/papers/kelly-paper.pdf>. Acessoem 16 novembro 2012.

LOEST, S. R. **UM SISTEMA DE BACKUP COOPERATIVO TOLERANTE A INTRUSÕES BASEADO EM REDES P2P**. Curitiba: PUC-PR, 2007.67 p. Dissertação(Mestrado)- Programa de Pós-Graduação em Informática Aplicada.

LUA, K. CROWCROFT, J., PIAS, M., SHARMA, R., LIM, S.**A Survey and Comparison of Peer-to-Peer Overlay Network Schemes**.2004. Disponível em<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.109.6124>. Acesso em 19 novembro 2012.

MAIBAUM, N.; MUNDT, T. **JXTA**: A Technology Facilitating Mobile Peer-To-Peer Networks. 2002. Disponível em <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1166946>. Acesso em 15 novembro 2012.

RODRIGUES, R., DRUSCHEL, P. **Peer-to-Peer Systems**. 2010. Disponível em <http://www.cs.princeton.edu/courses/archive/spr11/cos448/web/docs/week11_optional1.pdf>. Acesso em 19 novembro 2012.

SUN MICROSYSTEMS. **JXTA v2.0 ProtocolsSpecification**. 2007. Disponível em <http://jxta.kenai.com/Specifications/JXTAProtocols2_0.pdf>. Acesso em 15 novembro 2012.

SUN MICROSYSTEMS. **Project JXTA v2.0**: Java™ Programmer'sGuide. 2003. Disponível em <http://pagesperso-systeme.lip6.fr/Sebastien.Monnet/PSIA/docs/JxtaProgGuide_v2.pdf>. Acesso em 15 novembro 2012.

SUN MICROSYSTEMS. **Project JXTA**: A Technology Overview. 2001. Disponível em <ftp://jano.ucauca.edu.co/cursos/Memorias/CITA2002/JITT-Documentos/Ubicua2/Fuentes/Papers/TechOverview.pdf>. Acesso em 15 novembro 2012.

SUN MICROSYSTEMS.**JXTA Java™ Standard Edition v2.5**: Programmers Guide.2007.Disponível em: <http://java.net/projects/jxta-

guide/sources/svn/content/trunk/src/guide_v2.5/JXSE_ProgGuide_v2.5.pdf> .Acesso em 15 novembro 2012.

TANENBAUM, A. S. **Sistemas Distribuídos – Princípios e Paradigmas**.2. Ed. São Paulo: Person. 2007.

TOMELIN, M. **TESTES DE SOFTWARE A PARTIR DA FERRAMENTA VISUAL TEST**. 2000. Disponível em <http://campeche.inf.furb.br/tccs/2001-I/2001-1marciotomelinvf.pdf>. Acesso em 17 junho 2013.

VITHOFT, M., H. **UM SERVIÇO DE GERENCIAMENTO SEGURO DE CONTEÚDOS P2P COMPARTILHADOS EM MANET**. Curitiba: PUC-PR, 2009. 117 p. Dissertação(Mestrado)- Programa de Pós-Graduação em Informática Aplicada.