UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE SOFTWARE P2P PARA COMPARTILHAMENTO ANÔNIMO DE ARQUIVOS

MARCELO FERREIRA DA SILVA

BLUMENAU
2013

2013/2-15

MARCELO FERREIRA DA SILVA

PROTÓTIPO DE SOFTWARE P2P PARA COMPARTILHAMENTO ANÔNIMO DE ARQUIVOS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Francisco Adell Péricas , M. Sc. - Orientador

BLUMENAU
2013

2013/2-15

PROTÓTIPO DE SOFTWARE P2P PARA COMPARTILHAMENTO ANÔNIMO DE ARQUIVOS

Por

MARCELO FERREIRA DA SILVA

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Francisco Adell Péricas, M. Sc. – Orientador, FURB
Membro:	Prof. Jhony Alceu Pereira, Esp. – FURB
Membro:	Prof. Paulo Fernando da Silva, M. Sc. – FURB

Blumenau, 05 de dezembro de 2013

Dedico este trabalho aqueles comprometidos e engajados na luta pela libertação da humanidade dos grilhões que a acorrentam.

AGRADECIMENTOS

Primeiramente à minha esposa e companheira Jessica, amor da minha vida que compartilha comigo a luta em todos os níveis e sentidos, não apenas pela sobrevivência, mas também pela transformação da sociedade.

Aos meus pais que com sua dedicação e suor batalharam durante toda a sua vida para me cuidar, educar e dar os estudos necessários que foram determinantes para eu alcançar os objetivos dessa jornada.

Aos meus irmãos e poucos amigos pela sua companhia em momentos de alegrias e tristezas.

Ao meu orientador Péricas, por toda a sua contribuição, experiência e conhecimento que foram fundamentais para conclusão deste trabalho.

O saber quando não humaniza deprava. Refina o crime e torna mais degradante a covardia.

Mikhail Bakunin

RESUMO

Este trabalho apresenta o desenvolvimento de um protótipo que permite compartilhar arquivos anonimamente. O trabalho discorre sobre o uso da plataforma Freenet, a qual disponibiliza um sistema de armazenamento e recuperação de informação distribuída por meio de conexões entre computadores em uma arquitetura *Peer-to-Peer*. Apresenta um estudo do resultado obtido da implementação de uma estratégia de pesquisa por determinados arquivos. Além disso, é disponibilizada uma interface de usuário que permite publicar, replicar e recuperar arquivos compartilhados.

Palavras-chave: Rede de computadores. *Peer-to-Peer*. Privacidade. Anonimato. Freenet.

ABSTRACT

This paper presents the development of a prototype that allows to share files anonymously. The paper discusses the use of Freenet platform, which provides a system for storing and retrieving information distributed through connections between computers in an architecture Peer-to-Peer. Presents a study of the results obtained from the implementation of a research strategy for certain files. Also, available is a user interface that allows you to publish, replicate and retrieve shared files.

Key-words: Computer network. Peer-to-Peer. Privacy. Anonymity. Freenet.

LISTA DE ILUSTRAÇÕES

Figura 1 - Típica requisição realizada pelo Freenet 18
Figura 2 - Diagrama de casos de uso 23
Quadro 1 - Caso de uso UC01 24
Quadro 2 - Caso de uso UC02 24
Quadro 3 - Caso de uso UC03 25
Quadro 4 - Caso de uso UC04 25
Figura 3 - Diagrama de classes da camada *view* do *plugin* 27
Figura 4 - Diagrama de classes da camada *core* do *plugin* 28
Figura 5 - Diagrama de sequência da interação do usuário com o protótipo 29
Figura 6 - Ferramenta para exportar ao FRED o jar do *plugin* desenvolvido 31
Quadro 5 - Método responsável por realizar *upload* do arquivo 32
Quadro 6 - Método responsável por inserir determinado arquivo 32
Quadro 7 - Método responsável por gerar uma chave de referência 33
Quadro 8 - Método responsável por realizar *download* do arquivo 33
Quadro 9 - Implementação do método responsável por buscar arquivo 34
Quadro 10 - Método responsável por realizar pesquisa por arquivos 34
Quadro 11 - Método responsável por realizar a pesquisa por metadados 35
Quadro 12 - Método responsável por baixar determinado arquivo metadata 35
Figura 7 - Tela de configuração do modo de segurança do Freenet 37
Figura 8 - Tela de configuração da segurança física do *cache* 37
Figura 9 - Tela de configuração para o nome do nó 38
Figura 10 - Tela de configuração do tamanho armazenamento dos dados 38
Figura 11 - Tela de configuração do tipo de conexão com a Internet 39
Figura 12 - Tela de configuração de *plugins* não-oficiais 39
Figura 13 - Tela com a lista de *plugins* carregados no Freenet 40
Figura 14 - Tela modal para carregamento do arquivo 41
Figura 15 - Tela de pesquisa por arquivos compartilhados 42
Figura 16 - Tela com determinado arquivo baixado 43
Quadro 13 - Características do trabalho desenvolvido e dos correlatos 45

LISTA DE SIGLAS

CHK – *Content-Hash Key*

DNS – *Domain Name System*

FCP – *Freenet Client Protocol*

FProxy – *Freenet Proxy*

FRED – *Freenet REference Daemon*

HTTP – *HyperText Transfer Protocol*

ISOC – *Internet SOCiety*

ISP – *Internet Service Provider*

KSK – *Keyword-Signed Key*

P2P – *Peer-to-Peer*

RF – Requisito Funcional

RNF – Requisito Não-Funcional

SSK – *Signed-Subspace Key*

TCP/IP – *Transmission Control Protocol/Internet Protocol*

UML – *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO 12
1.1 OBJETIVOS DO TRABALHO 13
1.2 ESTRUTURA DO TRABALHO 13
2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 REDES PEER-TO-PEER 15
2.2 PRIVACIDADE E ANONIMATO 16
2.3 FREENET 17
2.3.1 *File Keys* 18
2.3.2 Tabela de roteamento 19
2.3.3 Armazenamento dos dados 19
2.4 TRABALHOS CORRELATOS 20
2.4.1 Gnutella 20
2.4.2 Napster 21
3 DESENVOLVIMENTO 22
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 22
3.2 ESPECIFICAÇÃO 22
3.2.1 Diagrama de casos de uso 23
3.2.1.1 Configurar nó em uma rede de computadores 23
3.2.1.2 Realizar *upload* de arquivos 24
3.2.1.3 Realizar *download* de arquivos 25
3.2.1.4 Pesquisar arquivos compartilhados 25
3.2.2 Diagramas de classes 26
3.2.2.1 Classes da camada *view* do *plugin* 26
3.2.2.2 Classes da camada *core* do *plugin* 27
3.2.3 Diagramas de sequência 28
3.3 IMPLEMENTAÇÃO 30
3.3.1 Técnicas e ferramentas utilizadas 30
3.3.1.1 Montagem do ambiente de desenvolvimento 30
3.3.1.2 Implementação da classe `FilesharingCore` 31
3.3.1.3 Implementação da classe `FilesharingSearch` 34
3.3.2 Operacionalidade da implementação 36

3.3.2.1 Configuração do nó na rede 36
3.3.2.2 Realização de *upload* de arquivos 40
3.3.2.3 Pesquisa por arquivos compartilhados 41
3.3.2.4 Realização de *download* de arquivos 42
3.4 RESULTADOS E DISCUSSÃO 43
3.4.1 Comparativo com os trabalhos correlatos 44
4 CONCLUSÕES 46
4.1 EXTENSÕES 47
REFERÊNCIAS BIBLIOGRÁFICAS 48

1 INTRODUÇÃO

O aumento da vigilância por atores estatais e corporativos cada vez mais ameaça a liberdade de expressão na sociedade contemporânea. Com avanço tecnológico dos computadores pessoais e o advento da Internet, o compartilhamento de informações pessoais torna-se objeto de interesse destes atores quando se trata da censura e supressão da privacidade (CLARKE et al., 2002).

A rede mundial de computadores, Internet, surgiu por meio da interconexão da ARPANET, NSFNET e EBONE somente em 1990. Contudo, a sua arquitetura se desenvolveu desde a década de 60 com pesquisas do Departamento de Defesa dos Estados Unidos, até culminar com a especificação e implementação do *Transmission Control Protocol/Internet Protocol* (TCP/IP) e do *Domain Name System* (DNS) (PÉRICAS, 2010, p. 37).

Por definição, um dispositivo estará conectado a Internet quando executa os protocolos TCP/IP e tem um endereço de IP fixo ou temporário. A sua topologia ou estrutura de interconexão é hierárquica, onde as estações dos usuários são conectadas a um provedor local de acesso (*Internet Service Provider* - ISP) (PÉRICAS, 2010, p. 38).

A conectividade por meio da Internet potencializa a disponibilidade de informações para todos os seus usuários, permitindo assim, a liberdade de expressão. No entanto, acredita-se que a Internet, por si só, não pode ser controlada. Porém, por intermédio dos ISPs, é possível controlar e censurar grande parcela da rede, onde os usuários dos serviços disponíveis por ela são consideravelmente mais expostos do que anônimos (CLARKE et al., 2010).

Além disso, alguns organismos internacionais coordenados pela *Internet SOCiety* (ISOC) respondem pela manutenção e evolução da Internet, bem como questões relacionadas à censura, liberdade de expressão e propriedade intelectual (PÉRICAS, 2010, p. 38). O problema reside quanto ao controle externo da Internet, cuja forma de acesso a ela por meio das ISPs, organismos privados ou estatais, podem censurar ou até mesmo reter informações.

Para responder a estas contradições de caráter social, faz-se necessário conceber e desenvolver tecnologias capazes de garantir o direito livre de se expressar. Contudo, é relativamente recente a pesquisa e desenvolvimento de sistemas que visam a liberdade de expressão no compartilhamento de informações na Internet por meio da privacidade e do

anonimato. Além disso, a garantia da privacidade e do compartilhamento anônimo da informação em uma rede de computadores flexível e topologicamente hierárquica, como é a Internet, corresponde a um nível considerável de complexidade tecnológica.

Diante desta problemática, este trabalho disponibiliza implementações que visam garantir a liberdade de expressão por meio da privacidade e do anonimato no compartilhamento de informações na Internet. Para tal, isso deve ocorrer tanto para autores quanto para leitores de informações compartilhadas por meio da publicação, replicação ou recuperação de arquivos.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é pesquisar e desenvolver um protótipo de aplicação para compartilhamento anônimo de arquivos em uma arquitetura de rede descentralizada *Peer-to-Peer* (P2P).

Os objetivos específicos do trabalho são:

a) disponibilizar a conexão entre computadores em uma arquitetura P2P;
b) disponibilizar o compartilhamento anônimo e distribuído de arquivos utilizando as implementações básicas fornecida pela plataforma do Freenet (sistema de armazenamento e recuperação de informação distribuída);
c) disponibilizar um aplicativo para compartilhar arquivos anonimamente.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está desenvolvido em quatro capítulos, sendo que o segundo capítulo apresenta a fundamentação teórica referente aos conceitos e técnicas adotadas.

O terceiro capítulo trata do desenvolvimento do protótipo de compartilhamento anônimo de arquivos, contendo os requisitos e a especificação por meio dos diagramas de caso de uso, classe e sequência. Em seguida, prossegue com o detalhamento da implementação, resultados e problemas encontrados.

Ao final, no quarto capítulo são apresentadas as conclusões finais sobre o trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A seguir são explanados conceitos e técnicas adotados por este trabalho referentes às redes *Peer-to-Peer*, privacidade e anonimato, além do Freenet, um sistema de armazenamento de arquivos distribuídos anonimamente. Em seguida são abordados trabalhos que possuem algumas características afins.

2.1 REDES PEER-TO-PEER

Define-se *Peer-to-Peer* (P2P) como uma arquitetura de rede distribuída onde há compartilhamento de recursos de hardware entre os nós participantes, recursos esses necessários para compartilhar serviços e conteúdos disponibilizados pela rede (KELLERER, 1998 apud SILVA, 2003). Para que seja considerada puramente P2P, Kellerer (1998 apud SILVA, 2003) ainda afirma que esta rede deve ser "[...] em primeiro lugar P2P conforme a definição anterior e em segundo lugar que qualquer entidade terminal da rede possa ser removida sem que a rede sofra qualquer perda em termos de serviço."

A partir deste conceito de rede P2P percebe-se que a descentralização da arquitetura elimina a dependência de um servidor central entre os nós participantes. No entanto, isso acarreta uma série de novos problemas. Entre eles, Wilson (2002, p. 11) destaca a troca de mensagens sem a sobrecarga da rede, compartilhamento de arquivos de forma segura e anônima e incentivo no compartilhamento de recursos.

Abordando a cada um destes problemas em específico, surgem variantes de compartilhamento de arquivos P2P, incluindo Morpheus, software desenvolvido pela Streamcast (2002), a qual oferece recursos de pesquisa com base em metadados. Também Mojo (2000), que usa uma moeda artificial para impor a partilha de recursos, denominada Mojo. Além disso, há o Freenet (2000), que provê de forma descentralizada, arquivos protegidos por criptografia forte contra adulteração por meio da privacidade e anonimato (WILSON, 2002, p. 11).

2.2 PRIVACIDADE E ANONIMATO

Segundo Albuquerque e Ribeiro (2002, p. 1), segurança da informação deve ser analisada basicamente sob os aspectos da confidencialidade, integridade e disponibilidade. A primeira visa impedir que usuários não autorizados acessem determinada informação. A segunda tem por objetivo indicar se a informação compartilhada está consistente ou inalterada. Finalmente, a terceira procura garantir que o sistema realize uma tarefa solicitada sem falhas.

Contudo, dentre estes três aspectos principais, há a privacidade. Albuquerque e Ribeiro (2002, p. 2) definem também privacidade como sendo a "capacidade de um sistema de manter incógnito um usuário, impossibilitando a ligação direta da identidade do usuário com as ações por este realizadas." Por exemplo, o processo eleitoral brasileiro exige que não haja qualquer elo entre o eleitor e seu voto, caracterizando-o assim, como secreto ou privado.

A privacidade pode ser compreendida de diversas formas ou níveis. Isso irá variar de acordo com a necessidade de segurança do sistema. Conforme Albuquerque e Ribeiro (2002, p. 207), as seguintes características a compõe:

a) invisibilidade: demais usuários não têm como saber quem está usando determinado recurso do sistema;
b) não-rastreamento: ações não-interligadas à identidade de determinado usuário;
c) pseudônimo: recurso utilizado na necessidade de responsabilização do usuário. Este será autenticado, porém será identificado com um pseudônimo;
d) anonimato: não-identificação efetiva do usuário utilizada em sistemas onde a privacidade é bastante forte e as informações não são confidenciais.

A partir disso, surgem as questões de como garantir a privacidade no compartilhamento de informações não confidenciais em um sistema; e mais, em um sistema em que as informações deverão estar distribuídas e descentralizadas. Com o objetivo de solucionar estes problemas, há o Freenet, que é um sistema descentralizado de armazenamento e recuperação de arquivos que visa o compartilhamento das informações por meio da invisibilidade, não-rastreamento e anonimato dos usuários.

2.3 FREENET

Para Clarke et al. (2002), há um senso comum de que os mecanismos de anonimato podem ser utilizados por criminosos com objetivo de evitar a aplicação da lei. Como Freenet é projetado para compartilhar conteúdo para o mundo - não tão útil para organizações criminais - ele não é particularmente atraente para tais fins. De qualquer modo, a comunicação digital anônima é simplesmente uma ferramenta, como telefones ou correio postal, e que podem tanto ser utilizados eticamente como não (CLARKE et al., 2002).

Deste modo, segundo Clarke et al. (2002), o objetivo geral do Freenet é prover:

a) privacidade para os produtores de informação, consumidores e titulares;
b) resistência à censura de informação;
c) alta disponibilidade e confiabilidade através da descentralização;
d) armazenamento e roteamento eficiente, escalável e adaptável.

Para isso, a arquitetura do Freenet é baseada em uma rede P2P de nós, os quais armazenam e recuperam arquivos de dados. É mantido em cada nó um local próprio para armazenamento de dados que disponibiliza a leitura e escrita para a rede, bem como uma tabela de roteamento dinâmico contendo os endereços dos nós vizinhos (CLARKE et al., 2000).

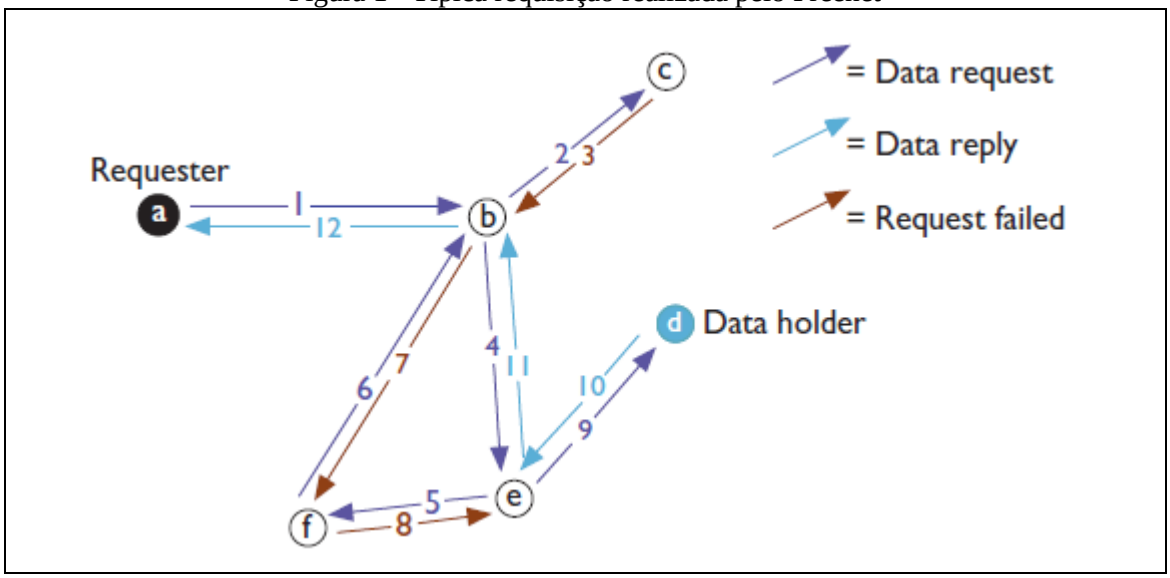
Os arquivos são armazenados ou inseridos na rede com um endereço ou chave associado. A chave é utilizada tanto para decidir em qual local da rede armazenar o arquivo e para autenticá-lo quando for transferido. O nó que desejar acessar o arquivo deverá enviar uma consulta de recuperação, a qual executará uma busca em profundidade. Quando o pedido encontra um nó que contém uma cópia do documento, todo o documento é devolvido através do caminho de pesquisa (CLARKE et al., 2010).

Deste modo, o sistema Freenet é uma ampla rede distribuída na Internet. Nela há integração de vários níveis de redundância de dados, onde estes são espalhados em toda a topologia da Internet, o que torna mais difícil identificar a origem ou remover dados da rede (CLARKE et al., 2010).

Contudo, o Freenet ainda é uma tecnologia recente, cujas limitações não permitem o compartilhamento de qualquer arquivo, mas de apenas *freesites* (páginas web do Freenet) (FREENET, 2010). Além disso, não fornece um sistema útil de busca de arquivos, apenas uma biblioteca, que busca *freesites* baseados em índices centralizados (FREENET, 2010).

No Freenet, para cada nó são disponibilizados dois serviços básicos à rede: armazenamento de dados e roteamento. O roteamento é gerenciado pela tabela de roteamento onde os nós são associados às *file keys*, já o dado é armazenado juntamente com a *file key* correspondente (SKOGH, 2006). Na Figura 1 é apresentada uma típica requisição no Freenet, onde ela se move por meio da rede, regride de um nó sem saída (etapa 3) e dá uma volta (etapa 7) até localizar o arquivo desejado.

Figura 1 - Típica requisição realizada pelo Freenet



Fonte: Clarke et al., 2002.

2.3.1 *File Keys*

Para executar ações como identificar, encriptar e encaminhar documentos, o Freenet utiliza chaves assimétricas por meio de três tipos diferentes: *Keyword-Signed Key* (KSK), *Signed-Subspace Key* (SSK) e C*ontent-Hash Key* (CHK) (CLARKE et al., 2002). Ao criar um par de chaves assimétricas, a inserção utiliza uma *string* descritiva do documento. Em seguida, o documento é assinado com a chave privada e encriptado com essa *string* como chave, a qual é então distribuída para que outros sejam capazes de solicitar e decriptar o documento (SKOGH, 2006).

2.3.2 Tabela de roteamento

As rotas conhecidas para os *file keys* são armazenadas em uma tabela de roteamento disponibilizada em cada nó. Caso o nó esteja envolvido em uma solicitação ou inserção com sucesso, ele torna-se ciente do nó que solicitou ou inseriu o documento (SKOGH, 2006).

O Freenet implementa o algoritmo *Least Recently Used* (LRU) devido ao tamanho da tabela de roteamento ser limitado (SKOGH, 2006). Isso é, se a tabela está completa e uma nova rota é adicionada, o item menos recente é substituído pelo mais recente, ou seja, as rotas obsoletas e inválidas são substituídas pelas mais populares que permanecem na tabela.

2.3.3 Armazenamento dos dados

Para o armazenamento dos dados, novamente é implementado o algortimo LRU da mesma forma que a tabela de roteamento. Porém, ao invés de ser limitado pelo número de rotas, limita-se pelo tamanho de todos os documentos armazenados. Caso a capacidade de armazenamento de dados limitada pelo nó estiver esgotada, então vários documentos pequenos serão substituídos por um único documento, onde os documentos mais populares (amplamente distribuídos na rede) se manterão mais do que o menos populares (SKOGH, 2006).

O nó que armazena determinado documento não possui a chave (*string* descritiva do documento) para decriptá-lo. No entanto, apesar de todos os documentos serem encriptados individualmente, há uma *file key* no armazenamento de dados associada a cada documento, permitindo, durante o roteamento ou requisição, a identificação do documento armazenado (SKOGH, 2006).

2.4 TRABALHOS CORRELATOS

Pode-se encontrar trabalhos correlacionados à proposta. Apesar de existirem trabalhos correlatos mais recentes como XMPP (2004) e não tão recente como Bittorrent (2001), os sistemas mais conhecidos e semelhantes são o Gnutella (2000) e o Napster (1999). Ambos implementam em larga escala o compartilhamento de espaço em disco por meio da arquitetura de rede P2P.

2.4.1 Gnutella

É um sistema descentralizado P2P de tal forma que permite que os nós compartilhem recursos deles com outros nós conectados a rede. De acordo com Silva (2003), a comunicação na rede Gnutella ocorre somente com os nós que o usuário autorizou a conexão, e as requisições são transmitidas de par a par. Nestas requisições, HTTP é o protocolo utilizado. Com isso, a pesquisa por um arquivo ocorre da seguinte forma:

a) um nó A transmite a requisição para um nó B que por sua vez a armazena e caso não possa atender re-encaminha para outros nós, inclusive um nó C;
b) se o nó C puder atender, então ele envia a resposta para o nó B que verifica no registro quem solicitou e encaminha para o nó A.

Gnutella é um protocolo simples, o qual apenas define como uma mensagem é passada de um nó ao outro e como cada nó o interpreta. As limitações básicas são a grande quantidade de mensagens trocadas, a latência decorrente e o fato de ser impossível alcançar a rede de forma universal, pois o nó estará limitado a encontrar informações que estejam restritas ao seu horizonte de busca, o que é feita em paralelo (SILVA, 2003).

2.4.2 Napster

Possibilita que qualquer nó compartilhe arquivos MP3 com nós conectados ao servidor Napster. Devido a isso, a sua arquitetura é considerada P2P híbrido. Além disso, seus servidores lidam com a transferência dos arquivos entre os clientes, porém não os armazenam (SILVA, 2003).

Por meio do Napster, uma vez o sistema conectado à Internet, ele passa a se comunicar com o servidor central. Este, por sua vez, concentra o diretório de todos os computadores clientes conectado a ele. Caso um usuário requeira determinado arquivo, ele faz uma requisição ao servidor central, o qual verifica em seu diretório se existe um arquivo que corresponda ao requerimento feito pelo cliente. A partir disso, o servidor então envia ao usuário todos os arquivos encontrados, caso existirem. Depois o usuário escolhe o arquivo desejado da lista correspondente ao requerimento e tenta estabelecer uma comunicação direta com o local onde se localiza o arquivo escolhido (DURANTE, 2004).

Uma solução que torna eficiente a troca de arquivos MP3 por meio do Napster são os metadados que contém informações sobre artista, título, etc (SILVA, 2003).

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as etapas do desenvolvimento do protótipo Filesharing. São abordados os principais requisitos, a especificação, a implementação e por fim os resultados e discussão dos problemas encontrados.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O protótipo da aplicação para compartilhamento anônimo de arquivos deverá:

a) compartilhar arquivos em uma rede de computadores (Requisito Funcional - RF);
b) realizar o *upload* de arquivos compartilhados por meio de uma interface de usuário (RF);
c) realizar o *download* de arquivos compartilhados por meio de uma interface de usuário (RF);
d) possibilitar a pesquisa de arquivos compartilados por meio de uma interface de usuário (RF);
e) garantir a privacidade e anonimato no compartilhamento de arquivos (RF);
f) disponibilizar o compartilhamento em uma arquitetura de rede descentralizada P2P baseado na plataforma do Freenet (Requisito Não-Funcional - RNF);
g) disponibilizar uma interface *web* de usuário (RNF);
h) ser implementado na linguagem de programação Java (RNF).

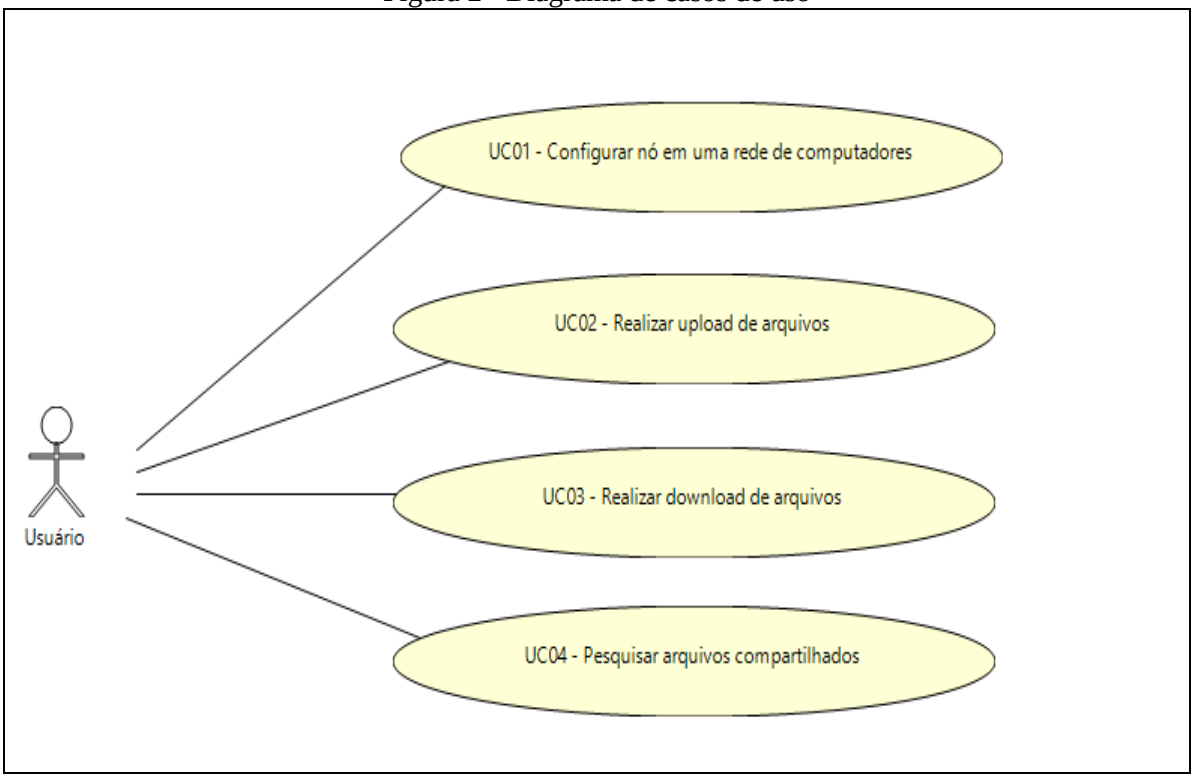
3.2 ESPECIFICAÇÃO

A especificação deste trabalho foi desenvolvida utilizando os conceitos da *Unified Modeling Language* (UML) por meio da ferramenta Papyrus e ObjectAid, o primeiro para os diagrama de caso de uso e sequência, e o segundo para o diagrama de classe. Ambos são *plugins* para o ambiente de desenvolvimento Eclipse.

3.2.1 Diagrama de casos de uso

Nesta seção são apresentados os casos de uso utilizados pelo ator `Usuário`. Esse irá interagir com as funcionalidades do protótipo apresentadas no diagrama da Figura 2.

Figura 2 - Diagrama de casos de uso



3.2.1.1 Configurar nó em uma rede de computadores

Este caso de uso especifica o que o ator `Usuário` deverá configurar em seu nó para conectá-lo a uma rede disponbilizada pelo Freenet e instalar o *plugin* Filesharing, protótipo desenvolvido por este trabalho. Como pré-condição ele deve ter o Freenet instalado em seu computador. A instalação pode ser realizada pelo instalador do Freenet. Contudo, é possível baixar seus códigos fontes, compilar e executar por meio da plataforma do Java. Ao executá-lo pela primeira vez, o sistema irá criar as configurações básicas padrões, as quais poderão ser personalizadas por meio da interface *Freenet Proxy* (FProxy) disponibilizada pelo próprio Freenet. O Quadro 1 apresenta o detalhamento deste caso de uso.

Quadro 1 - Caso de uso UC01

UC01 - Configurar nó em uma rede de computadores	
Descrição	Para compartilhar arquivos anonimamente o usuário deve configurar seu nó em uma rede *Peer-to-Peer* disponibilizada pelo Freenet.
Pré-condição	O usuário deve ter instalado a plataforma Freenet em seu computador.
Cenário principal	1. o usuário executa o Freenet; 2. o usuário seleciona o modo de segurança Opennet ou Darknet; 3. o usuário fornece um nome ao nó; 4. o usuário seleciona a quantidade de espaço do disco; 5. o usuário informa o tipo de conexão com a Internet (ADSL, VDSL, etc.); 6. o usuário instala o *plugin* Filesharing no Freenet; 7. o usuário conecta a um nó semente ou a outros nós amigos.
Pós-condição	O usuário deve estar conectado a outros nós em uma rede de computadores *Peer-to-Peer*.

3.2.1.2 Realizar *upload* de arquivos

Este caso de uso especifica de que forma o ator `Usuário` interagirá com o sistema para carregar ou inserir arquivos na rede Freenet. Além disso, o Quadro 2 demonstra como os arquivos metadados serão inseridos para facilitar a pesquisa.

Quadro 2 - Caso de uso UC02

UC02 - Realizar *upload* de arquivos	
Descrição	Carrega ou insere arquivos na rede Freenet, bem como seu respectivo metadados para facilitar a pesquisa.
Pré-condição	O usuário deve ter instalado o *plugin* Filesharing no Freenet e conectado a um nó semente ou a outros nós amigos.
Cenário principal	1. o usuário executa o botão "Carregar arquivo" 2. o sistema abrirá uma tela modal, a qual permite carregar arquivos de qualquer formato; 3. o usuário informa o tipo de chave de referência ao arquivo (CHK ou SSK); 4. o usuário informa o caminho do arquivo a ser carregado; 5. o usuário executa o botão "Carregar arquivo"; 6. o sistema insere, na rede Freenet, o arquivo fornecido pelo usuário com sua respectiva chave de referência; 7. o sistema insere, na rede Freenet, um arquivo metadata com sua respectiva chave de referência do tipo KSK. O metadados possui informações do arquivo fornecido pelo usuário.
Pós-condição	O arquivo deverá ter sido carregado ou inserido na rede Freenet, bem como o respectivo arquivo com os metadados.

3.2.1.3 Realizar *download* de arquivos

Este caso de uso especifica o processo de *download* de arquivos da rede Freenet que o ator `Usuário` irá realizar por meio do protótipo. Para isso ele deverá ter pesquisado o arquivo desejado possuindo sua chave de referência. Detalhes podem ser conferidos no Quadro 3.

Quadro 3 - Caso de uso UC03

UC03 - Realizar *download* de arquivos	
Descrição	Para baixar arquivos o usuário deve ter iniciado e configurado seu nó na rede Freenet e possuir as chaves de referências para os arquivos.
Cenário principal	1. o usuário pesquisa ou fornece uma chave de referência para o arquivo desejado; 2. o sistema baixa, da rede Freenet, o arquivo da chave de referência fornecida.
Pós-condição	O arquivo deverá ter sido baixado da rede Freenet e salvo em um diretório padrão do nó.

3.2.1.4 Pesquisar arquivos compartilhados

Este caso de uso descreve a funcionalidade de pesquisa a ser realizada pelo ator `Usuário`. Como pré-condição o arquivo deve ter sido inserido pelo próprio protótipo. O Quadro 4 apresenta os detalhes desse processo.

Quadro 4 - Caso de uso UC04

UC04 - Pesquisar arquivos compartilhados	
Descrição	Pesquisa metadados de arquivos compartilhados.
Pré-condição	Para efetuar pesquisa o arquivo deve ter sido carregado pelo *plugin* Filesharing. Além disso, o nó deve estar conectado a uma rede Freenet.
Cenário principal	1. o usuário informa no campo de pesquisa o nome do arquivo desejado; 2. o sistema irá buscar arquivos metadados que contém o nome do arquivo informado pelo usuário. Para isso, irá utilizar a chave do tipo KSK. 3. o sistema baixa o conteúdo do metadado, caso encontre o arquivo conforme passo 2.
Pós-condição	Retorna lista com arquivos metadados. Por exemplo: nome do arquivo, chave para baixar o arquivo, caminho onde será baixado, etc.

3.2.2 Diagramas de classes

Nesta seção são demonstradas as principais classes desenvolvidas para o protótipo. Contudo, algumas classes auxiliares foram omitidas, porém isso não compromete o entendimento dos diagramas. A interface disponibilizada pelo Freenet permite implementar tanto integração com o *core* quanto com a *view* do sistema. Para isso, na implementação do protótipo foi utilizado o padrão de projeto *Model View Controller* (MVC). Desta forma, primeiramente é apresentado a camada *view* implementada pelo protótipo que se integrará com o Freenet, em seguida, é exposto a camada do *core*, a qual implementa as funcionalidades especificadas nos casos de uso.

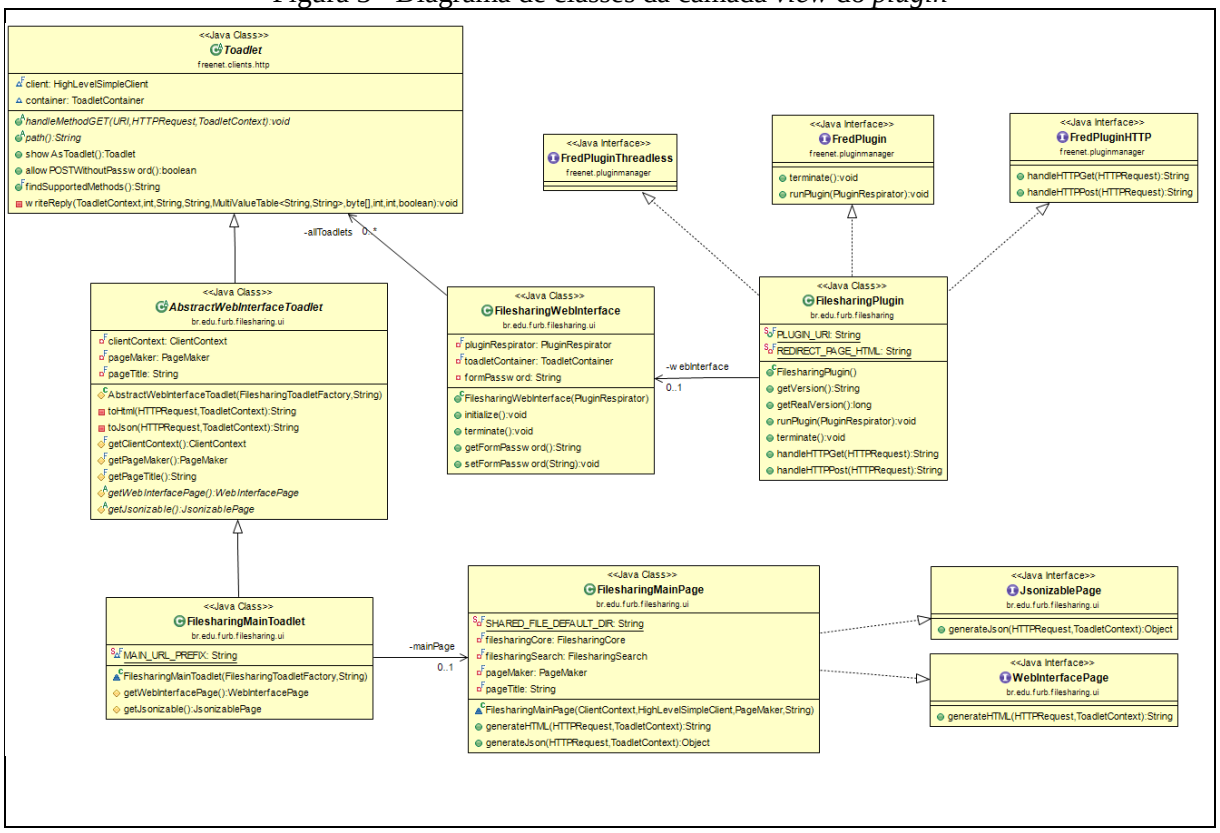
3.2.2.1 Classes da camada *view* do *plugin*

Nesta seção são apresentadas as classes da camada *view* do MVC implementado pelo protótipo. A classe `FilesharingPlugin` é responsável por integrar com o Freenet por meio da implementação da interface `FredPlugin`. Essa é realizada utilizando o padrão de projeto Plugin, de forma que o Freenet instancia o *plugin* referenciado em um arquivo de configuração (no caso do Freenet, arquivo `.manifest`), o qual contém a versão do *plugin* e o caminho completo da classe do *plugin* que será instanciada.

Para integrar com a parte *web* do sistema, o Freenet disponibiliza a interface `FredPluginHTTP`, a qual permite atender requisições *post* e *get* do protocolo *HyperText Transfer Protocol* (HTTP). Além disso, há a classe `Toadlet`, a qual funciona de forma semelhante à classe `Servlet` do Java, porém com algumas funcionalidades a mais implementadas. Para isso, foi implementada a classe `FilesharingWebInterface`, cuja função é registrar os *servlets* responsáveis por atender determinada requisição *web*, incluindo o *servlet* implementado pela classe `FilesharingMainToadlet`. Essa classe tratará requisições de geração de *HyperText Markup Language* (HTML) e *JavaScript Object Notation* (JSON).

A Figura 3 apresenta o diagrama de classe da camada *view* implementada pelo *plugin* do protótipo.

Figura 3 - Diagrama de classes da camada *view* do *plugin*



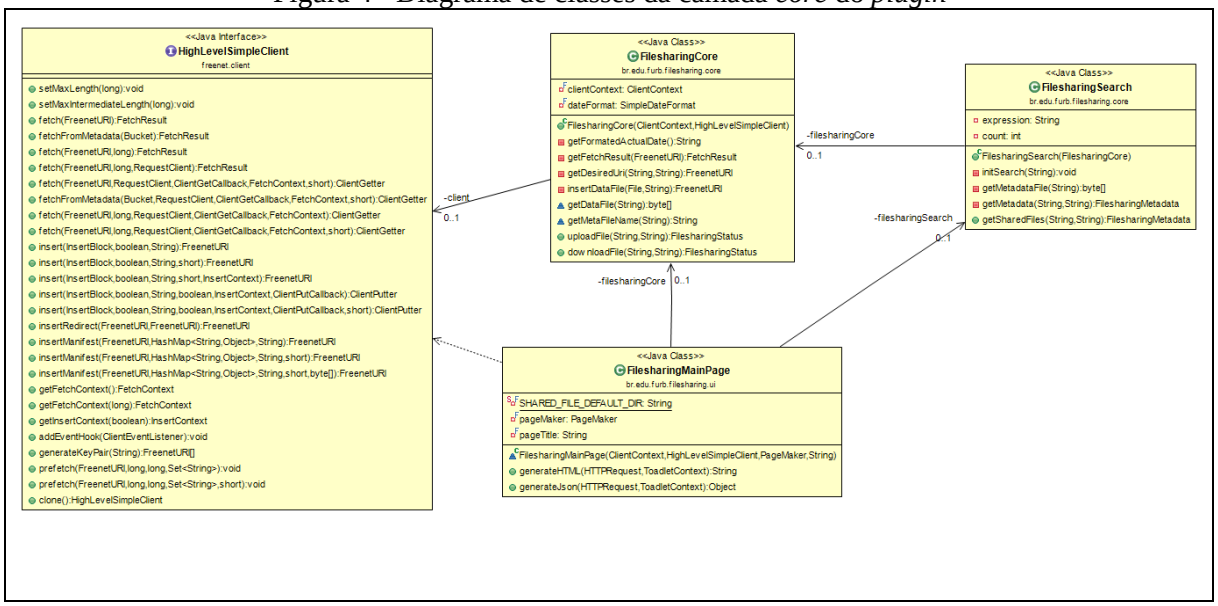
3.2.2.2 Classes da camada *core* do *plugin*

Nesta seção são apresentadas as classes da camada *core* do MVC implementado pelo protótipo. A classe `FilesharingCore` implementa a integração com o *client* do Freenet por meio da interface `HighLevelSimpleClient`. Essa disponibiliza funções básicas de inserção e recuperação de arquivos armazenados na rede do Freenet.

Para atender as requisições web de *upload*, *download* e pesquisa de arquivos, a classe `FilesharingMainPage` utiliza o *core* do *plugin*. Desta forma, ela possui assossiação com a classe `FilesharingCore`, que basicamente possui a responsabilidade de centralizar a comunicação com o *client* do Freenet, permitindo assim, o tratamento de solicitações de inserção e recuperação de arquivos. Há também a classe `FilesharingSearch`, a qual se associa à classe `FilesharingCore` para atender solicitações de pesquisa.

A Figura 4 apresenta o diagrama de classe da camada *core* implementada pelo *plugin* do protótipo.

Figura 4 - Diagrama de classes da camada *core* do *plugin*



3.2.3 Diagramas de sequência

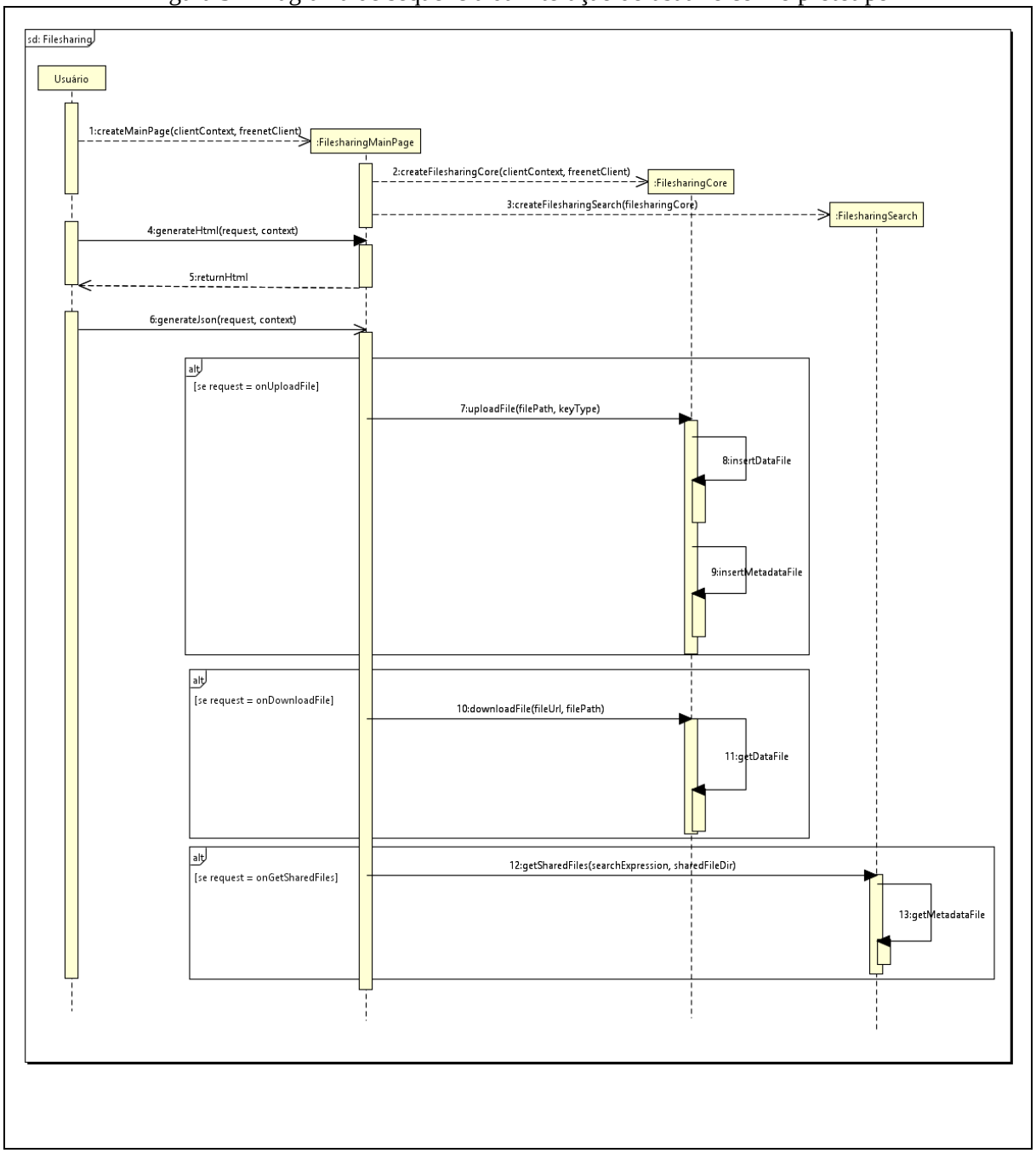
Esta seção apresenta o diagrama de sequência, o qual especifica a interação do ator `Usuário` com o protótipo. Nele é demonstrado o fluxo de mensagens entre a parte *view* e *core* desenvolvidos utilizando padrão de projeto MVC.

Inicialmente o ator `Usuário` inicializa a execução do protótipo Filesharing, o qual desencadeia a instanciação do *plugin*, que passa a aguardar requisições *web*. A classe `FilesharingMainPage` é criada, bem como as classes do *core* `FilesharingCore` e `FilesharingSearch`. Em seguida, por meio do sistema, o ator `Usuário` solicita a geração do HTML da página principal do protótipo. Esse retorna a página contendo referências para os códigos implementados em JavaScript e *Cascading Style Sheets* (CSS).

Finalizado o processo inicial, conforme as solicitações do ator, o sistema passa a gerar requisições HTTP assíncronas utilizando *Asynchronous Javascript and* XML (AJAX). Por meio dessas, caso a requisão for `onUploadFile` a mensagem de carregamento aciona inserção do arquivo na rede bem como o seu respectivo arquivo com metadados. Caso contrário, se a requisisão for `onDownloadFile` a mensagem de descarregamento aciona a recuperação do arquivo da rede utilizando a sua respectiva chave de referência. Já, se a requisição for `onGetSharedFiles` a mensagem de busca aciona a pesquisa por arquivos da rede conforme a expressão fornecida pelo ator `Usuário`, o qual retornará a lista de seus respectivos arquivos

metatados. A Figura 5 apresenta o diagrama de sequência das trocas de mensagens do ator com a camada *core* do protótipo.

Figura 5 - Diagrama de sequência da interação do usuário com o protótipo



3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

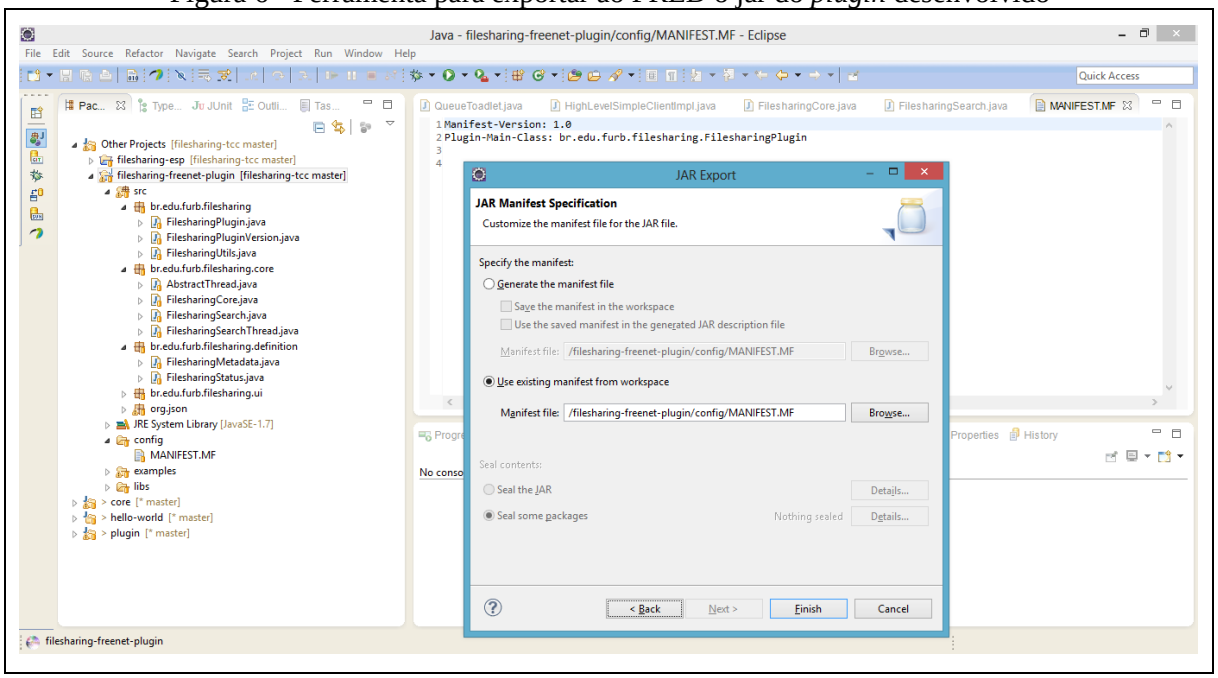
3.3.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento do protótipo a partir da especificação foi utilizada a linguagem de programação Java, bem como o ambiente de desenvolvimento Eclipse. A camada *web* foi implementada utilizando a linguagem de programação Javascript em conjunto com CSS e HTML. Além disso, a biblioteca OrgJson foi utilizada para transformar objetos *Plain Old Java Objects* (POJO) para notação JSON. Já a camada *core* foi implementada utilizando *Freenet REference Daemon* (FRED), biblioteca reponsável por integrar o *plugin* ao Freenet.

3.3.1.1 Montagem do ambiente de desenvolvimento

Para montar o ambiente de desenvolvimento é necessário baixar os códigos fontes do FRED em (FREENET, 2013). Após isso, também deve-se baixar as bibliotecas `freenet-ext.jar` e `bcprov.jar` em (FREENET, 2011a) para conseguir compilar o FRED. A Figura 6 demonstra como exportar o *plugin* desenvolvido para que seja possível instalar o mesmo no Freenet. Basicamente, é preciso gerar uma arquivo `.manifest`, o qual conterá os atributos `Manifest-Version` e `Plugin-Main-Class`. Esse último referencia o caminho completo da classe que implementa a interface `FredPlugin`, que no caso deste protótipo, é `br.edu.furb.filesharing.FileSharingPlugin`.

Figura 6 - Ferramenta para exportar ao FRED o jar do *plugin* desenvolvido



3.3.1.2 Implementação da classe `FilesharingCore`

Esta seção apresenta os principais métodos do *core* do protótipo. Encontra-se entre eles funções básicas para inserção e recuperação de arquivos da rede do Freenet. A classe que as implementam é a `FilesharingCore`. Ela é responsável por integrar com Freenet utilizando a interface `HighLevelSimpleClient`.

No Quadro 5 é demonstrado o método `uploadFile`. Conforme o diagrama de sequência apresentado na Figura 5, este método é responsável por inserir o arquivo na rede, bem como armazenar a chave de referência gerada em um arquivo metadata que também será inserido por ele. O nome do arquivo metadata é gerado (linhas 8 e 9) com base na data atual de inserção do arquivo, isso para facilitar a pesquisa em ordem de inserção. Diferentemente do arquivo original que é gerado com chave fornecida pelo ator `Usuário`, o arquivo metadata é inserido na rede com a chave do tipo KSK (linha 25). Ao final é retornado o *status* indicando que o *upload* ocorreu com sucesso (linha 29).

Quadro 5 - Método responsável por realizar *upload* do arquivo

```
01 public FilesharingStatus uploadFile(final String filePath, final
02 String keyType) throws Exception {
03
04       final File file = new File(filePath);
05       final FreenetURI desiredURI = this.insertDataFile(file, keyType);
06       final String insertedFileName = file.getName().split("\\.")[0];
07
08       final File metadataFile =
09             new File(this.getMetaFileName(insertedFileName));
10
11       try {
12             FileOutputStream fos = null;
13             try {
14                   fos = new FileOutputStream(metadataFile);
15                   final StringBuilder sb = new StringBuilder();
16                   sb.append("<filename>");
17                   sb.append(file.getName());
18                   sb.append("\n");
19                   sb.append("<fileurl>");
20                   sb.append(desiredURI.toASCIIString());
21                   fos.write(sb.toString().getBytes());
22             } finally {
23                   Closer.close(fos);
24             }
25             this.insertDataFile(metadataFile, "KSK");
26       } finally {
27             metadataFile.delete();
28       }
29       return FilesharingStatus.UPLOADED;
30 }
```

O Quadro 6 apresenta o método `insertDataFile`. Este método faz uso da interface `HighLevelSimpleClient` para inserir o arquivo na rede do Freenet. Para isso, ele gera uma chave de referência (linhas 6 e 7) de acordo com tipo recebido no parâmetro `keyType`. E, nas linhas 12 e 13 é possível notar que é gerado um `filenameHint` caso a chave for do tipo CHK, isso ocorre devido uma exigência do *client* do Freenet. No final da inserção ele retorna uma chave de referência derivada da gerada.

Quadro 6 - Método responsável por inserir determinado arquivo

```
01 private FreenetURI insertDataFile(final File file, final String
02 keyType) throws MalformedURLException, InsertException {
03       final Bucket data =
04             new FileBucket(file, true, false, false, false, false);
05       final FreenetURI desiredURI =
06             this.getDesiredUri(file.getName(), keyType);
07       final InsertBlock insert =
08             new InsertBlock(data, null, desiredURI);
09       final boolean isCHKOnly = false;
10       final String filenameHint =
11             desiredURI.isCHK() ? file.getName() : null;
12       this.client.insert(insert, isCHKOnly, filenameHint);
13       return desiredURI.deriveRequestURIFromInsertURI();
14 }
```

No Quadro 7 é apresentada a função `getDesiredUri`. Esta função tem o papel de, a partir de um nome do arquivo gerar uma chave de refência conforme o tipo especificado no parâmetro `keyType`.

Quadro 7 - Método responsável por gerar uma chave de referência

```
01 private FreenetURI getDesiredUri(final String fileName, final String
02 keyType) throws MalformedURLException {
03
04       final FreenetURI desiredURI = new FreenetURI(keyType + "@");
05
06       if (desiredURI.isSSK()) {
07             final InsertableClientSSK insertableKey =
08       InsertableClientSSK.createRandom(this.clientContext.random, "");
09             return insertableKey.getInsertURI().setDocName(fileName);
10       }
11
12       if (desiredURI.isKSK()) {
13             final ClientKSK insertableKey = ClientKSK.create(fileName);
14             return insertableKey.getURI();
15       }
16
17       return desiredURI;
18 }
```

Já o Quadro 8 demonstra o método `downloadFile`. Este método tem a função de, a partir da chave de referência especificada no parâmetro `fileUrl`, baixar o arquivo na caminho especificado em `filePath`. Impliscitamente este método invoca a função `getFetchResult` por meio da linha 5. Ao final é retornado o *status* indicando que o *download* ocorreu com sucesso (linha 11).

Quadro 8 - Método responsável por realizar *download* do arquivo

```
01 public FilesharingStatus downloadFile(final String fileUrl, final
02 String filePath) throws Exception {
03       FileOutputStream fos = null;
04       try {
05             final byte[] dataFile = this.getDataFile(fileUrl);
06             fos = new FileOutputStream(filePath);
07             fos.write(dataFile);
08       } finally {
09             Closer.close(fos);
10       }
11       return FilesharingStatus.DOWNLOADED;
12 }
```

Finalmente, o Quadro 9 apresenta a função `getFetchResult`. Esta função tem a responsabilidade de, por meio da interface `HighLevelSimpleClient`, obter um `FetchResult` do arquivo requisitado pelo parâmentro `freenetUri`. Ao analisá-la percebe-se que há implementação de tolerância a falhas que pode vir a ocorrer ao baixar o arquivo da rede do

Freenet, onde a busca pelo arquivo ocorre dentro de um laço e, caso ocorra `FetchException` e `newURI` está setada, é realizada uma nova tentativa de busca ao arquivo (linha 10).

Quadro 9 - Implementação do método responsável por buscar arquivo

```
01 private FetchResult getFetchResult(FreenetURI freenetUri) throws
02 FetchException {
03       while (true) {
04             try {
05                   return this.client.fetch(uri);
06             } catch (final FetchException e) {
07                   if (e.newURI == null) {
08                         throw e;
09                   }
10                   uri = e.newURI;
11             }
12       }
13 }
```

3.3.1.3 Implementação da classe `FilesharingSearch`

Esta seção apresenta as principais funções necessárias para realizar a pesquisa por determinado arquivo. A classe que as implementam é `FilesharingSearch`, a qual faz uso da classe `FilesharingCore` ao utilizar funções necessárias para recuperação de arquivos.

No Quadro 10 é demonstrado o método `getSharedFiles`. Conforme o diagrama de sequência apresentado na Figura 5, este método é responsável por pesquisar arquivos metadatas na rede do Freenet, isso para obter as chaves de referência, as quais serão utilizadas para baixar o arquivo desejado pelo ator `Usuário`.

Quadro 10 - Método responsável por realizar pesquisa por arquivos

```
01 public FilesharingMetadata getSharedFiles(final String
02 searchExpression, final String sharedFileDir) throws Exception {
03       if (searchExpression.trim().isEmpty() || this.count == 0) {
04             this.initSearch(searchExpression);
05             return new FilesharingMetadata(searchExpression);
06       }
07       if (!(this.expression.equals(searchExpression))) {
08             this.initSearch(searchExpression);
09       }
10       final FilesharingMetadata metadata =
11             this.getMetadata(this.expression, sharedFileDir);
12       if (metadata.isExists()) {
13             metadata.setSearchExpression(this.expression);
14             metadata.setStop(this.count-- == 0);
15       }
16       return metadata;
17 }
```

O Quadro 11 apresenta o método `getMetadata`. Este método tem a função de, interpretar o conteúdo de um determinado arquivo metadata recuperado por meio do parâmetro `searchExpression` (da linha 11 até a 20).

Quadro 11 - Método responsável por realizar a pesquisa por metadados

```
01 private FilesharingMetadata getMetadata(final String searchExpression,
02 final String sharedFileDir) throws IOException {
03
04       final byte[] metadataFile =
05             this.getMetadataFile(searchExpression);
06
07       if (metadataFile == null) {
08             return new FilesharingMetadata(searchExpression);
09       }
10
11       final String metadataContent = new String(metadataFile);
12       final BufferedReader bufferedReader =
13             new BufferedReader(new StringReader(metadataContent));
14       try {
15             final String fileName =
16                   bufferedReader.readLine().replace("<filename>", "");
17             final String fileUrl =
18                   bufferedReader.readLine().replace("<fileurl>", "");
19             return new FilesharingMetadata(fileName, fileUrl,
20 sharedFileDir + "\\" + fileName);
21       } finally {
22             Closer.close(bufferedReader);
23       }
24 }
```

Finalmente, o Quadro 12 apresenta o método `getMetadataFile`. Este método faz uso da classe `FilesharigCore` para recuperar o arquivo metadata da rede do Freenet. Impliscitamente por meio da função `getMetaFileName` é obtido o nome do arquivo metadata. Nota-se que para recuperar o arquivo é utilizada a chave de referência do tipo KSK.

Quadro 12 - Método responsável por baixar determinado arquivo metadata

```
01 private byte[] getMetadataFile(final String searchExpression) {
02    try {
03       final String metaFileName =
04             this.filesharingCore.getMetaFileName(searchExpression);
05
06       return this.filesharingCore.getDataFile("KSK@" + metaFileName);
07
08    } catch (final Exception e) {
09       return null;
10    }
11 }
```

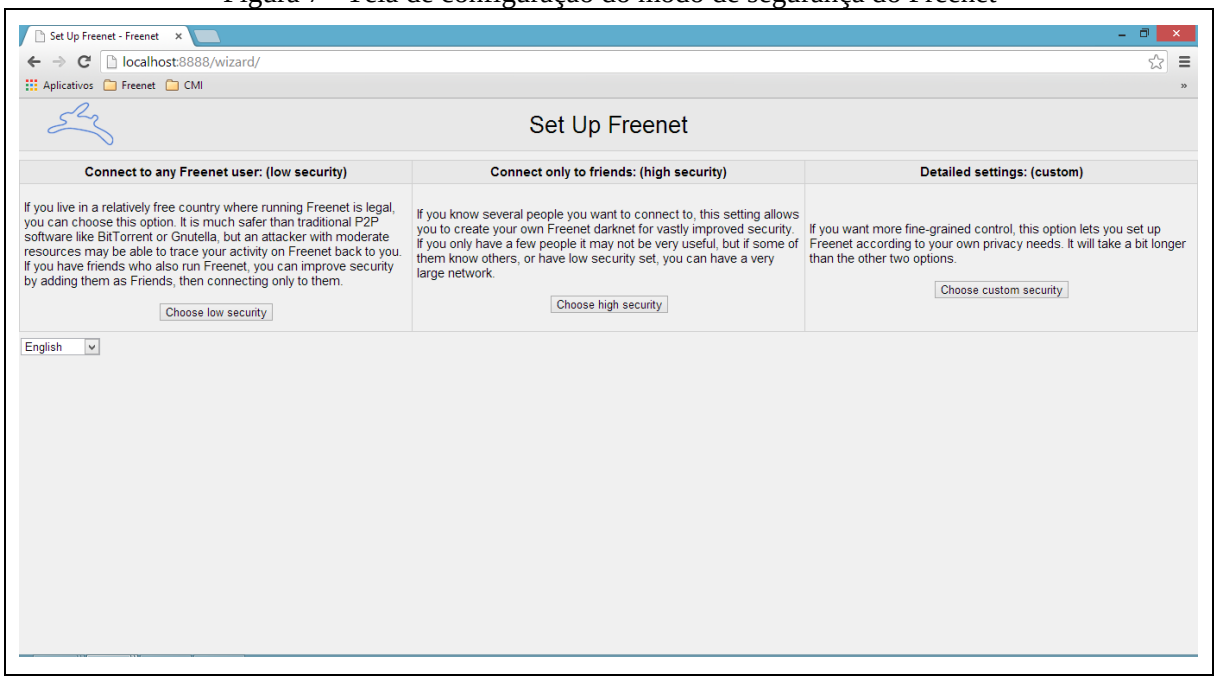
3.3.2 Operacionalidade da implementação

Esta seção tem por objetivo apresentar a operacionalidade da implementação. Isso é realizado mostrando a interação do ator `Usuário` com o protótipo por meio de estudo de caso. Primeiramente é apresentado a configuração do nó na rede Freenet. Logo após, é demonstrado como realizar *uploads*, *downloads* e pesquisa por arquivos compartilhados.

3.3.2.1 Configuração do nó na rede

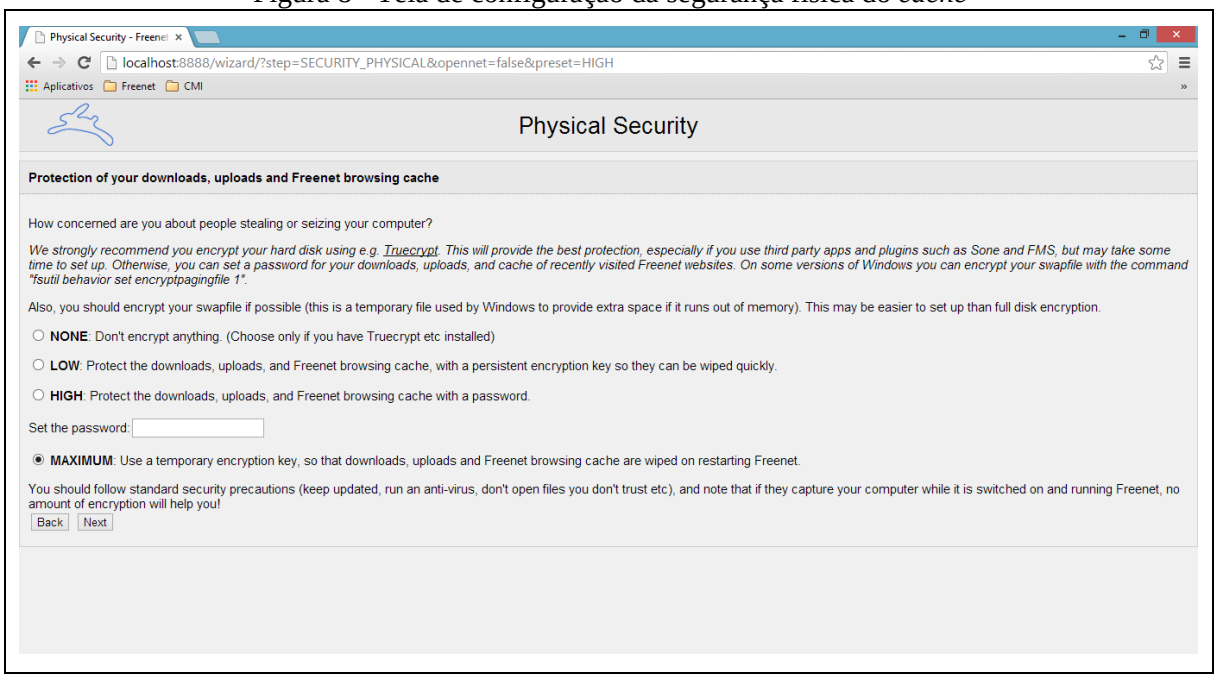
Para configurar o nó na rede é necessário ter o Freenet instalado e iniciado no computador. Conforme demonstrado na Figura 7, uma *wizard* de configuração incial é apresentada ao ator `Usuário` quando ele acessar em seu *browser* a *Uniform Resource Locator* (URL) padrão `http://localhost:8888` (a porta varia de acordo com o que foi configurado na instalação do Freenet). São disponibilizados os modos de conexão OpenNet (conecta-se a qualquer outro nó) e DarkNet (conecta-se apenas a determinado nó de confiança). A conexão OpenNet é mais vulnerál a ataques de rede, já o modo DarkNet permite uma conexão com a mais alta segurança de resistência à censura, ao ataque *Denial of Service* (DoS) e ao anonimato. Geralmente, o modo OpenNet é utilizado para conectar-se aos *seednodes* (nós sementes) para que se possa experimentar o Freenet antes de solicitar conexão aos nós confiáveis. Há ainda, a opção de personalização do modo de segurança, a qual irá variar de acordo com a necessidade privacidade do ator.

Figura 7 - Tela de configuração do modo de segurança do Freenet



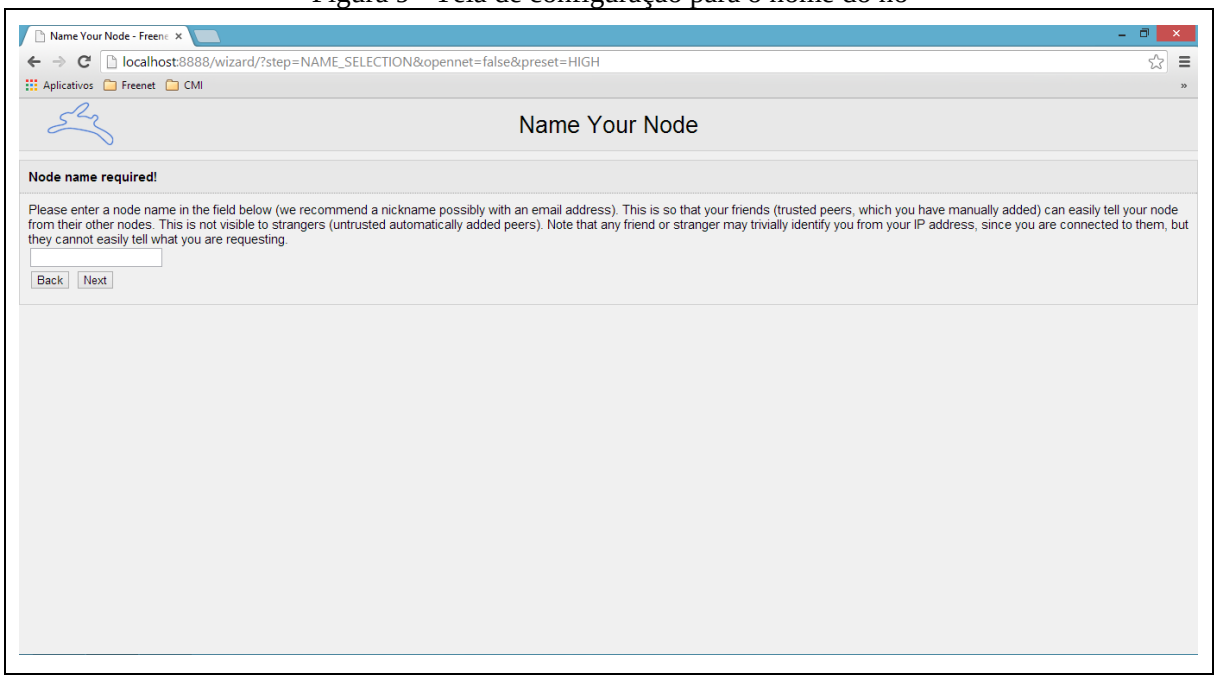
Na Figura 8 é apresentada a tela de configuração para proteção de *caches* mantidos pelo Freenet com relação aos *downloads* e *uploads*. É disponibilizada a opção `None`, a qual não encripta *cache* algum. Há ainda, a opção `Low` que é utilizada para encriptar o *cache* com uma chave de encriptação persistente. Além disso, há a opção `High` e `Maximum`, a primeira protege o *cache* com senha e a segunda remove o *cache* a cada reinicialização do Freenet.

Figura 8 - Tela de configuração da segurança física do *cache*



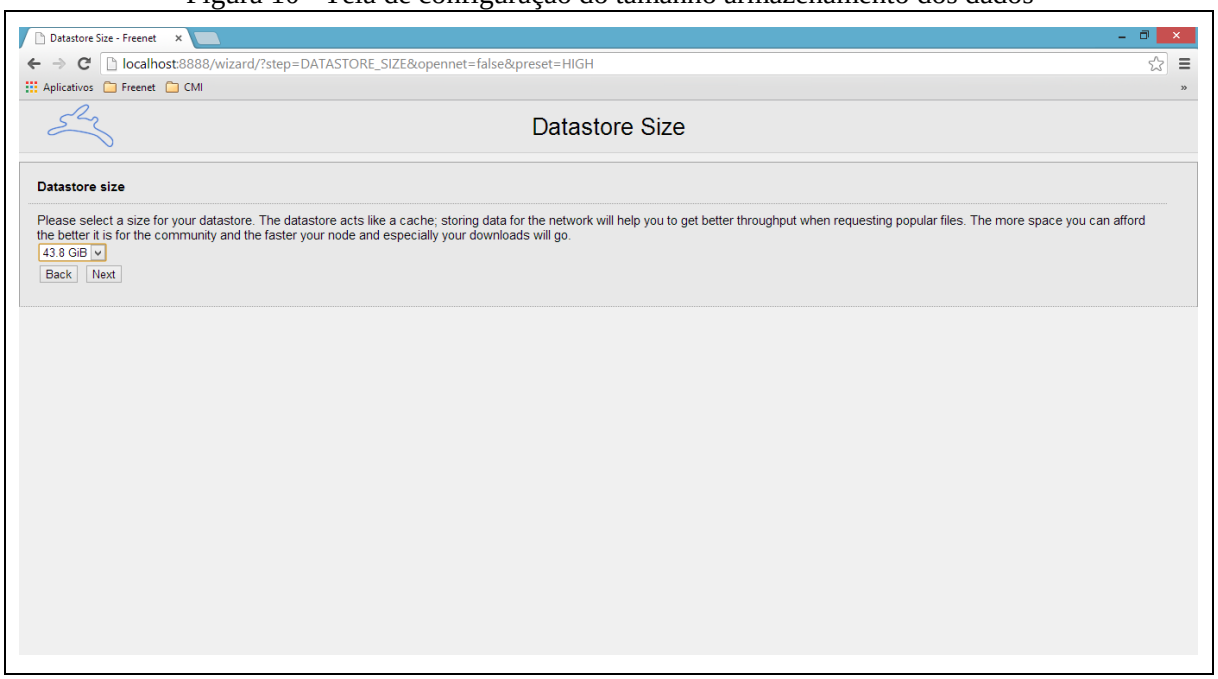
A Figura 9 apresenta a tela de configuração para o nome do nó. Recomenda-se que o nome seja um endereço de e-mail, a qual somente nós confiáveis conhecem.

Figura 9 - Tela de configuração para o nome do nó



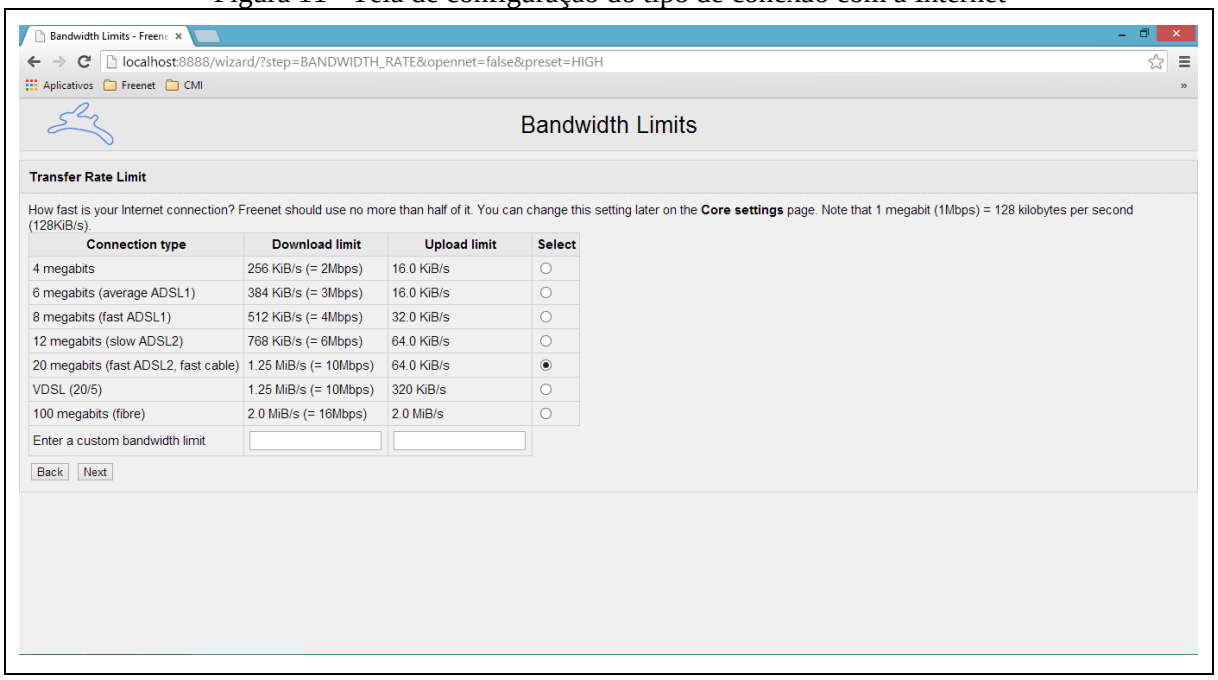
A tela de configuração do tamanho de espaço no disco a ser disponibilizado pelo nó é apresentada na Figura 10. Nela o ator `Usuário` precisa fornecer a quantidade do armazenamento de dados a ser disponibilizado à rede do Freenet. Devido a isso, o Freenet não garante a permanência de arquivos impopulares ou antigos na rede.

Figura 10 - Tela de configuração do tamanho armazenamento dos dados



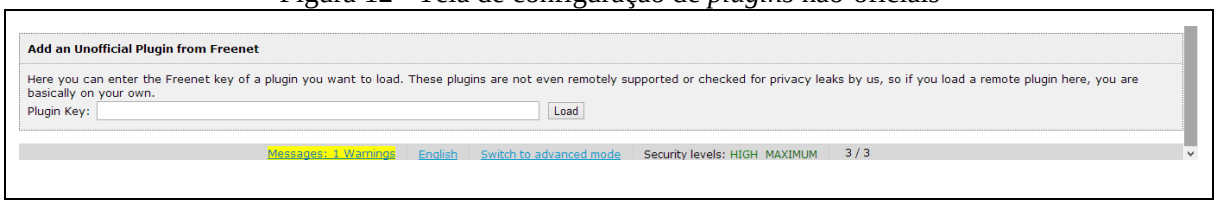
O ator Usuário ainda pode configurar o limite de taxa de transfência a ser utilizado pelo Freenet conforme a largura de banda da conexão com Internet do nó. A Figura 11 apresenta a tela de configuração do tipo de conexão com a Internet.

Figura 11 - Tela de configuração do tipo de conexão com a Internet



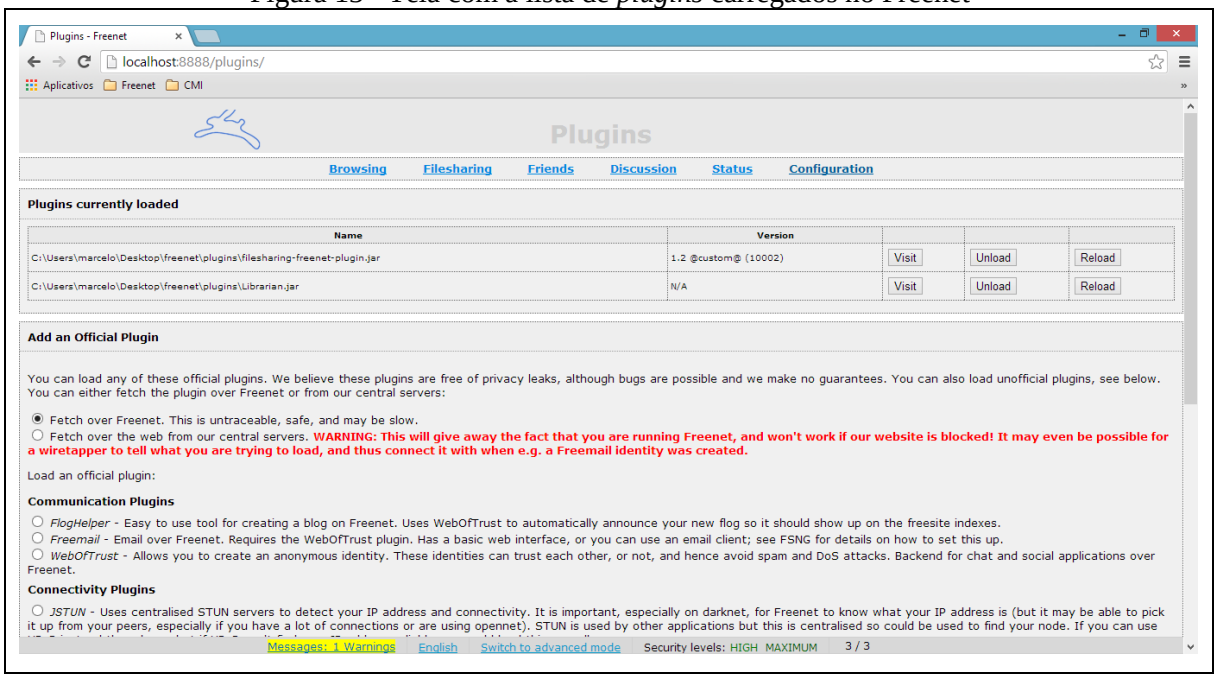
Já na Figura 12 é apresenta a tela onde o ator `Usuário` deve carregar *plugins* não-oficiais ao Freenet. É nessa tela que o *plugin* desenvolvido por este protótipo deverá ser carregado no Freenet conforme a exportação apresentada na Figura 6.

Figura 12 - Tela de configuração de *plugins* não-oficiais



Finalmente, a Figura 13 apresenta a tela onde é possível o ator Usuário visualizar os *plugins* carregados no Freenet. É por meio dela que o ator irá visitar os *plugins* instalados, inclusive o desenvolvido por este protótipo.

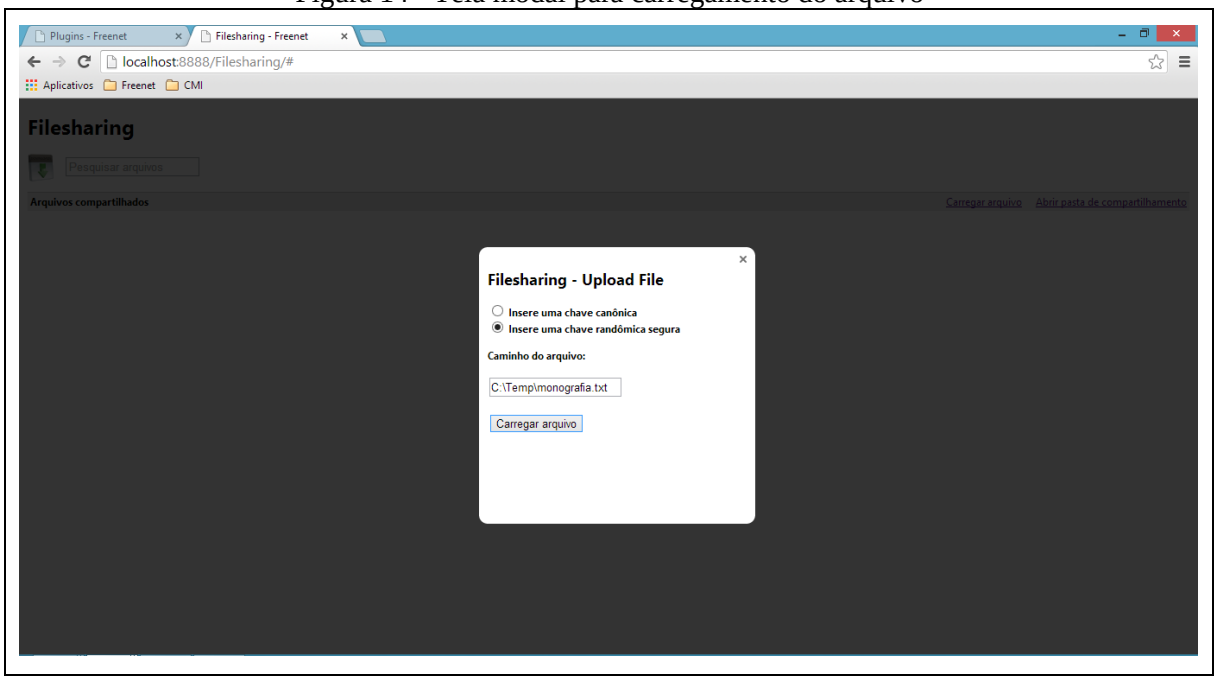
Figura 13 - Tela com a lista de *plugins* carregados no Freenet



3.3.2.2 Realização de *upload* de arquivos

Para realizar *upload* de um determinado arquivo, o ator `Usuário` deve clicar no botão `Carregar Arquivo`. Será aberta uma tela modal destinada a fornecer o local onde se encontra o arquivo e o tipo de chave de referência que a ser criada para ele. Esta chave pode ser canônica (CHK) ou randômica (SSK). A vantagem de utilizar a chave do tipo CHK é que será gerada a mesma para o mesmo arquivo. Já para o tipo SSK será gerada uma nova chave a cada inserção do mesmo arquivo, mas com a vantagem dela ser mais segura. Geralmente a chave CHK é utilizada para o modo OpenNet do Freenet e a SSK para o modo DarkNet. A Figura 14 apresenta a tela modal a ser mostrada ao ator.

Figura 14 - Tela modal para carregamento do arquivo



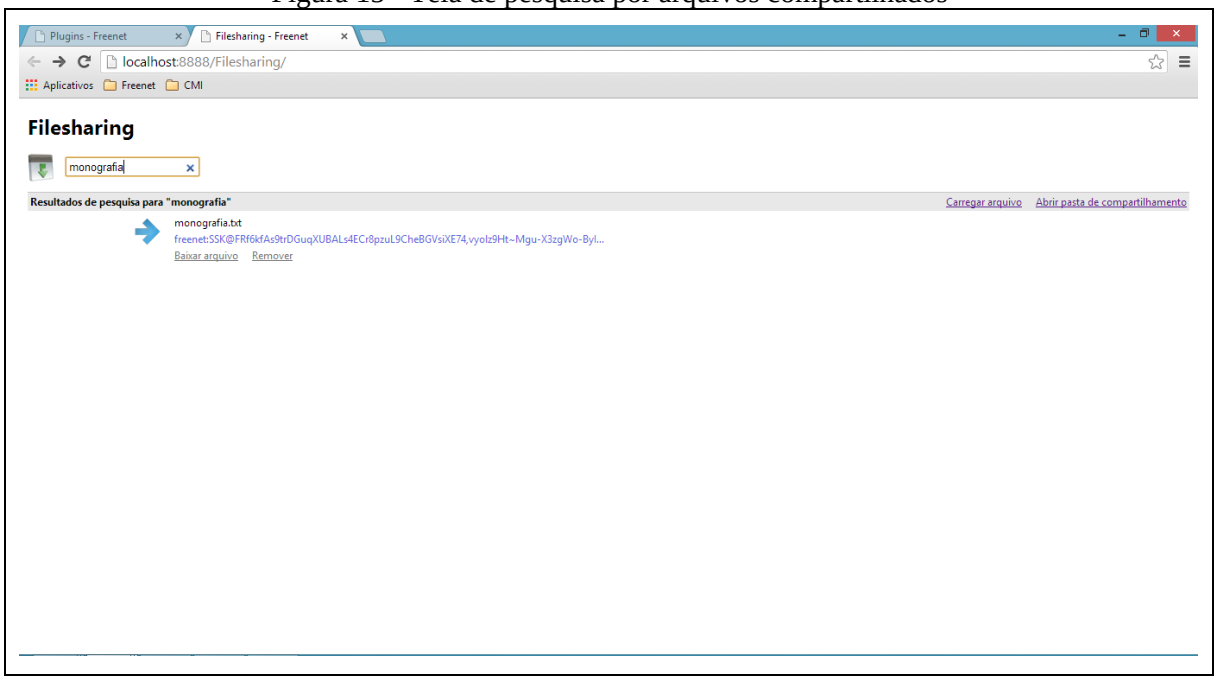
3.3.2.3 Pesquisa por arquivos compartilhados

Para pesquisar arquivos compartilhados na rede Freenet, o ator `Usuário` deve informar a expressão de pesquisa no campo de edição. Será efetuada uma busca por arquivos metadatas na rede Freenet que contenham a informação a ser pesquisada. Desta forma, somente arquivos compartilhados por nós utilizando este protótipo é que serão pesquisados. Isso, pois esse ao carregar o arquivo no Freenet, o insere em conjunto do arquivo metadata respeitando o seguinte formato:

a) `<filename>`: conterá o nome do arquivo e seu formato;
b) `<fileurl>`: conterá o tipo da chave SSK ou CHL seguido de @ e o valor da chave.

Cada arquivo metado é apresentado ao ator contendo a opção de baixar o arquivo referenciado ou de removê-lo da lista. A Figura 15 apresenta a tela a ser mostrada ao ator.

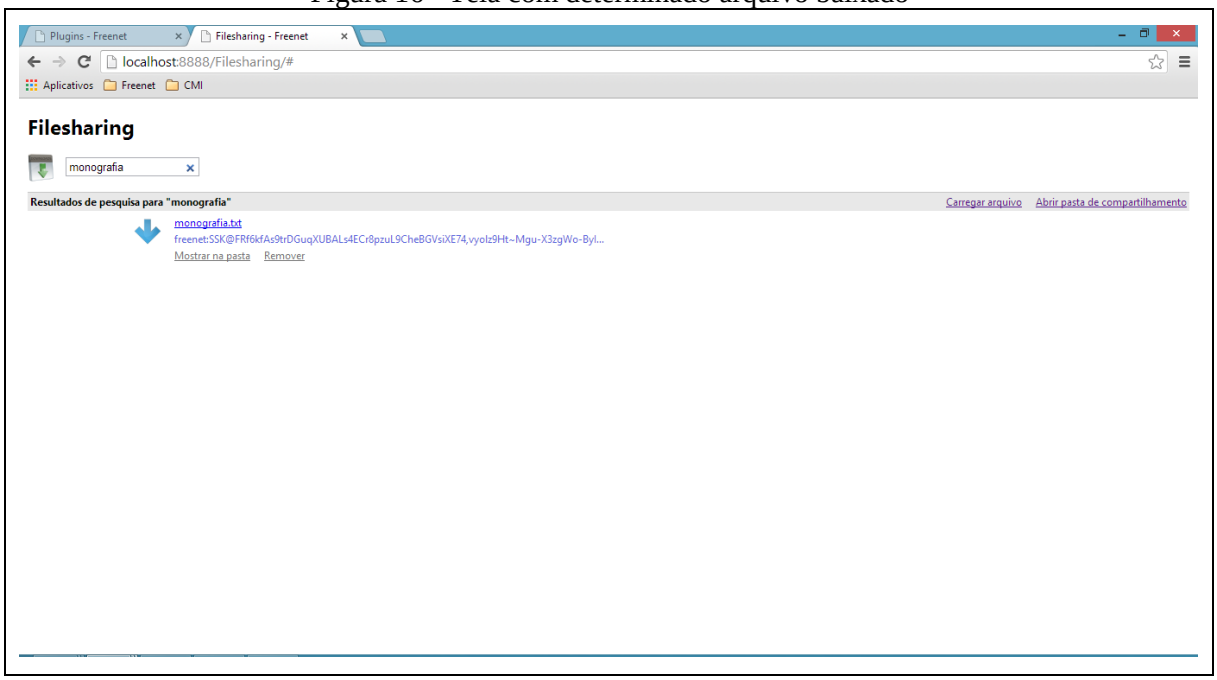
Figura 15 - Tela de pesquisa por arquivos compartilhados



3.3.2.4 Realização de *download* de arquivos

O *download* de arquivos compartilhados na rede Freenet pode ser realizado pelo ator `Usuário` após realizar a pesquisa por arquivos desejados. A pesquisa irá fornecer a chave de referência necessária para baixar o arquivo. Feito isso, o arquivo é armazenado no diretório padrão do computador do `Usuário`. Ao realizar o *download* ele terá a opção de abrir o arquivo por meio do *link* disponibilizado em seu título ou ainda de abrir o diretório padrão com o arquivo posicionado por meio do botão `Mostrar na pasta`. A Figura 16 apresenta a tela a ser mostrada ao ator.

Figura 16 - Tela com determinado arquivo baixado



3.4 RESULTADOS E DISCUSSÃO

O presente trabalho teve como objetivo pesquisar e desenvolver um protótipo de aplicação para compartilhamento anônimo de arquivos em uma arquitetura de rede descentralizada P2P. Além disso, apresentou uma implementação de pesquisa por determinados arquivos utilizando a estratégia de metadados contendo as informações básicas dos arquivos compartilhados.

O desenvolvimento foi baseado na plataforma do Freenet e não houveram mudanças relacionadas aos objetivos previamente definidos. Apesar das dificuldades encontradas devido à pouca documentação dessa plataforma, a depuração do *kernel* do Freenet possibilitou compreender sua arquitura interna. Isso viabilizou a integração do *plugin* desenvolvido no protótipo deste trabalho às peculiaridades de implementação relacionadas ao uso do Freenet.

Para o protótipo decidiu-se implementar um *plugin* na camada *web* disponibilizada pelo FProxy, já que a maioria dos usuários que navegam na Internet utilizam *browsers*. Desta forma, não foi utilizada a opção disponibilizada pelo Freenet em desenvolver um *client desktop*, o qual implementa o *Freenet Client Protocol* (FCP).

Para qualquer aplicativo com o objetivo de compartilhar arquivos é fundamental possuir a funcionalidade de pesquisa. Desta forma, inicialmente a ideia foi implementar um

recurso de pesquisa por arquivos compartilhados capaz de encontrar qualquer arquivo inserido no Freenet. Porém, devido a necessidade de metadados, a pesquisa passou a depender da inserção de um arquivo metadata em conjunto com o arquivo original. Com isso, para que a funcionalidade de pesquisa funcione no protótipo desenvolvido, os arquivos devem ter sido carregados ou inseridos pelo mesmo.

Outro detalhe a ser observado em relação ao recurso de pesquisa é que o arquivo metadata também deve ser inserido de forma a garantir a privacidade e o anonimato do usuário. Contudo, a chave de referência para o arquivo metadata não pode ser randômica. Para resolver esse problema, utilizou-se a chave do tipo KSK para inserção do arquivo metadata. Ela referencia o arquivo de acordo com o seu respectivo nome. A desvantagem de utilizar chaves KSK é que elas não são seguras com relação à *spam*, onde por meio dela, qualquer nó atacante pode inserir um arquivo com o mesmo nome com o intuito de desviar as requisões de um determinado arquivo metadata para o seu próprio arquivo que conterá informações falsas (FREENET, 2011b).

Por fim, percebeu-se problemas de desempenho relacionados à inserção do arquivo no Freenet, bem como à pesquisa por arquivos inexistentes ou que foram inseridos muito antigamente, ou ainda, que são arquivos impopulares. O baixo desempenho na inserção ocorre pois o Freenet insere o arquivo de modo em que ele é replicado aos diversos nós espalhados na rede. Já na pesquisa é devido a necessidade de realizar uma busca em profundidade em toda a rede caso não encontre os arquivos de interesse. Além disso, a tentativa de busca por determinado arquivo metadata ocorre a partir da data atual até as mais antigas. Contudo, o desempenho da pesquisa aumentaria de forma considerável se fosse implementado um *webcrawler* (motor de busca com o objetivo de rastrear arquivos na *web*) capaz de realizar buscas em *background* e criar índices para os arquivos metadatas encontrados na rede.

A seguir é discutido o comparativo deste trabalho com os trabalhos correlatos.

3.4.1 Comparativo com os trabalhos correlatos

Nesta seção são discutidas as principais características deste trabalho em comparação aos trabalhos correlatos. O Quadro 13 apresenta esta comparação.

Quadro 13 - Características do trabalho desenvolvido e dos correlatos

Principais Características / trabalhos correlatos	Gnutella	Napster	Protótipo Filesharing
Rede de computadores P2P	X	X	X
Rede P2P puramente decentralizada	X		X
Compartilhamento de arquivos	X	X	X
Compartilhamento anônimo de arquivos			X
Replicação dos arquivos na rede			X
Recurso de pesquisa		X	X
Recurso de pesquisa por metadados		X	X
Código fonte aberto			X

Em relação aos trabalhos correlatos, o protótipo desenvolvido Filesharing possui todas as suas principais características. O Quadro 17 demonstra que o principal diferencial é garantir a privacidade e anonimato no compartilhamento de arquivos. E, apesar da popularidade do Gnulella e Napster, esses sistemas não possuem o código fonte aberto. Em relação a implementação da rede decentralizada P2P o Gnutella é que mais se aproxima do Filesharing. No entanto, quando o assunto é recurso de pesquisa, apenas o Napster possui busca por arquivos metadatas além do Filesharing. Já quando se trata de replicação dos arquivos na rede, apenas o Filesharing possui essa característica, onde os arquivos são persistentes na rede e não no nó que o publicou, onde eles estão disponíveis mesmo quando seus autores não estão conectados.

4 CONCLUSÕES

Constatou-se que a privacidade e liberdade de expressão encontram-se ameaçadas na sociedade contemporânea, pois órgãos de vigilância estatais e corporativos tem aumentado cada vez mais o interesse por informações compartilhadas pela Internet (CLARKE et al., 2002). Apesar disso, há um senso comum de que os mecanismos de anonimato podem ser utilizados por criminosos com objetivo de evitar a aplicação da lei, porém a comunicação digital anônima é simplesmente uma ferramenta, como telefones ou correio postal, e que podem tanto ser utilizados eticamente como não (CLARKE et al., 2002).

Este trabalho atingiu seus objetivos específicos: o desenvolvimento de um protótipo de *software* permite o compartilhamento de informações, garantindo assim, a liberdade de expressão por meio da privacidade e anonimato. Para tal, disponibilizou a conexão entre computadores em uma arquitetura P2P, a qual possibilitou o compartilhamento anônimo e distribuído de arquivos utilizando a plataforma do Freenet. Além disso, também foi disponibilizada uma interface de usuário que permite publicar, replicar e recuperar arquivos compartilhados.

Pode-se dizer que o resultado obtido da implementação de uma estratégia de pesquisa por determinados arquivos metadatas atende os requisitos definidos. Contudo, recomenda-se incrementar essa implementação desenvolvendo um *webcrawler* com o objetivo de realizar buscas em *background* e criar índices para os arquivos metadatas encontrados na rede. Devido às vulnerabilidades do Freenet em relação à chaves do tipo KSK, sugere-se pesquisar uma outra alternativa mais segura de referenciar arquivos metadatas.

Em relação aos trabalhos correlatos, a principal diferença é que nenhum destes sistemas visa à privacidade e anonimato. Além disso, eles proporcionam apenas um serviço de compartilhamento de arquivos, porém não os armazenam. Isso é, os nós disponibilizam arquivos para os outros, mas não os replicam para armazenamento em outros nós. Nesta arquitetura os dados não são persistentes na rede, onde os arquivos apenas estão disponíveis quando seus autores estão conectados.

4.1 EXTENSÕES

No decorrer do desenvolvimento do trabalho foram identificados pontos de aprimoramento sugeridos abaixo:

a) implementar índices com base nos metadados e um *webcrawler* capaz de pesquisar metadados de forma assíncrona;
b) utilizar a biblioteca Apache Lucene para pesquisar palavras chaves nos metadados;
c) implementar um *plugin* para o Freenet capaz de recarregar o arquivo na rede com objetivo de mantê-lo disponível caso não seja muito popular;
d) implementar validações contra *spam* no caso de metadados inseridos pela chave KSK;
e) possibilitar o cancelamento de *downloads* interrompendo sua execução;
f) permitir o usuário configurar o diretório da pasta de compartilhamento onde os arquivos serão baixados;
g) desenvolver um protótipo de aplicativo *desktop* capaz de compartilhar arquivos anonimamente utilizando o FCP.

REFERÊNCIAS BIBLIOGRÁFICAS

ALBUQUERQUE, Ricardo; RIBEIRO, Bruno. **Segurança no desenvolvimento de software:** como garantir a segurança do sistema para seu cliente usando a ISSO/IEC. Rio de Janeiro: Campus, 2002.

BITTORRENT. **About Bittorrent**. San Francisco, USA, 2001. Disponível em: <http://www.bittorrent.com/intl/pt/company/about>. Acesso em: 16 maio 2013.

CLARKE, Ian et al. **Private communication through a network of trusted connections:** the dark Freenet. [S.l.], [2010]. Disponível em: <https://freenetproject.org/papers/freenet-0.7.5-paper.pdf>. Acesso em: 11 mar. 2013.

CLARKE, Ian et al. **Protecting free expression online with Freenet**. [S.l.], [2002]. Disponível em: <https://freenetproject.org/papers/freenet-ieee.pdf>. Acesso em: 11 mar. 2013.

CLARKE, Ian et al. **Freenet:** a distributed anonymous information storage and retrieval system. [S.l.], [2000]. Disponível em: <http://www.facweb.iitkgp.ernet.in/~niloy/COURSE/Autumn2010/UC/Resource/freenet1-big.pdf>. Acesso em: 11 mar. 2013.

DURANTE, Gabriel B. **Redes peer-to-peer**. Rio de Janeiro, [2004]. Disponível em: <http://www.gta.ufrj.br/grad/04_1/p2p/index.html>. Acesso em: 2 abr. 2013.

GNUTELLA forums. [S.l.], [2000]. Disponível em: <http://www.gnutellaforums.com>. Acesso em: 2 abr. 2013.

FREENET project. **Freenet:** the free network. [S.l.], [2000]. Disponível em: <https://www.freenetprojetct.org>. Acesso em: 11 mar. 2013.

_____. **Filesharing**. [S.l.], 2010. Disponível em: <https://wiki.freenetproject.org/Filesharing>. Acesso em: 17 maio 2013.

_____. **Freenet downloads**. [S.l.], 2011a. Disponível em: <https://downloads.freenetproject.org/alpha/>. Acesso em: 30 ago. 2013.

_____. **Understand Freenet**. [S.l.], 2011b. Disponível em: <https://downloads.freenetproject.org/alpha/>. Acesso em: 2 set. 2013.

_____. **Freenet reference daemon**. [S.l.], 2013. Disponível em: <https://github.com/freenet/fred-staging>. Acesso em: 10 ago. 2013.

MOJO nation. [S.l.], 2000. Disponível em: <http://sourceforge.net/projects/mojonation>. Acesso em: 13 abr. 2013.

NAPSTER network. [S.l.], [1999]. Disponível em: <http://www.napster.com>. Acesso em: 2 abr. 2013.

PÉRICAS, Francisco A. **Rede de computadores:** conceitos e arquitetura internet. 2. ed. Blumenau: Furb, 2010.

SILVA, William R. S. **Introdução às redes Peer-to-Peer (P2P)**. Rio de Janeiro, [2003]. Disponível em: <http://www.gta.ufrj.br/seminarios/semin2003_1/william/Gnutella.htm>. Acesso em: 2 abr. 2013.

SKOGH, Jonas H. **Improving Freenet performance by precedencial network partitioning and file mesh propagation**. [S.l.], 2006. Disponível em: <http://web.it.kth.se/~rassul/exjobb/rapporter/hansemil-jonas.pdf>. Acesso em: 8 set. 2013.

STREAMCAST. **Company overview**. [S.l.], 2002. Disponível em: <http://web.archive.org/web/20021010050856/http://www.streamcastnetworks.com/FullAbout.html>. Acesso em: 13 abr. 2013.

WILSON, Brendon J. **Project JXTA book**. Indianapolis, USA, [2002]. Disponível em: <http://www.brendonwilson.com/projects/jxta/pdf/JXTA.pdf>. Acesso em: 13 abr. 2013.

XMPP protocol. **XMPP standards foundation**. [S.l.], 2004. Disponível em: <http://xmpp.org/about-xmpp>. Acesso em: 16 maio 2013.