



Project 2 - Peer-to-peer file download

Turma: 2DA

Autores: João Santos 1140550;

Pedro Santos 1140386;

Luís Maia 1140776;

Professor: Joaquim Filipe Peixoto dos Santos

Unidade Curricular: Redes de Computadores

Índice

Introdução	2
Desenvolvimento	3
Descrição do Problema	3
Análise do problema.....	3
Protocolo UDP	4
Sockets de rede em UDP.....	4
Envio e recepção de datagramas UDP	4
Envio e recepção de datagramas.....	5
Tolerância a falhas	5
Envio em broadcast.....	6
Tamanho dos datagramas.....	6
Transações em múltiplos datagramas.....	6
Associação de endereços remotos	6
Servidores UDP	7
Protocolo TCP	7
Servidores TCP.....	8
Servidores TCP multi-processo.....	8
Resultados.....	9
Análise de Resultados.....	12
Primeira Thread.....	Erro! Marcador não definido.
Segunda Thread.....	Erro! Marcador não definido.
Terceira Thread.....	Erro! Marcador não definido.
Quarta Thread.....	Erro! Marcador não definido.
Conclusão.....	15
Referências Bibliográficas	16
Glossário	17

Introdução

No âmbito da unidade curricular de Redes de Computadores do curso de Engenharia Informática do Instituto Politécnico de Engenharia do Porto, foi proposto o desenvolvimento de uma aplicação de rede peer-to-peer, em que uma instância será capaz de descarregar ficheiros locais de outra instância. Será necessário o estudo aprofundado de alguns protocolos abordados nas aulas da unidade curricular. Para concretizar a aplicação, optou-se por programá-la em linguagem JAVA.

Este relatório visa explicar os conceitos mais importantes que tiveram de ser adquiridos para a realização da aplicação, bem como descrever a aplicação em si e apresentar os resultados obtidos.

Desenvolvimento

Descrição do Problema

Para o desenvolvimento da aplicação – Peer-to-peer file download- é necessário que haja várias instancias de uma única aplicação e que comuniquem entre si.

O seu principal objetivo é permitir que cada instancia descarregue ficheiros locais de outras instancias. Para isso, cada instancia deve, periodicamente, anunciar a lista de ficheiros que pretende partilhar, e deve ainda, construir a lista de todos os ficheiros partilhados pelas outras instancias. No final o utilizador deve selecionar o ficheiro que pretende descarregar.

Análise do problema

As aplicações de rede peer-to-peer têm características únicas e distintas. Uma das quais é de não seguir a arquitetura cliente-servidor, ou seja, não existe a aplicação de cliente nem de servidor, mas sim uma aplicação apenas capaz de desempenhar os dois papeis.

Cada instancia da aplicação publica a lista de ficheiros que pretende partilhar enviando datagramas UDP para o endereço de broadcast, e é enviado para uma porta fixa, que no nosso caso é a porta 32008. Desta porta, todas as instancias estão à espera de receber e recolher informação.

É possível trabalharmos num domino de broadcast por estarmos num ambiente de rede local. E um dos problemas a cuidar é que a própria instancia não disponha ao utilizador os ficheiros da sua própria publicação.

Cada anuncio de ficheiros partilhados deve ser enviado a cada 30 segundos. Como cada instancia está continuamente a receber anúncios a lista de ficheiros prontos a descarregar é permanentemente atualizada. Caso alguma outra instancia demore mais de 45 segundos a enviar o seu anuncio, então os ficheiros que estariam a ser partilhados por essa mesma instancia devem ser removidos da lista.

O descarregamento do ficheiro deve ser implementado através de uma conexão TCP feita com a instancia que publicou o ficheiro em causa. O numero da porta de cada instancia não é pré-definido, tendo que cada instancia usar uma porta atribuída dinamicamente. Esta atribuição dinâmica é feita no arranque da aplicação, e o numero da porta deve estar incluído na publicação dos ficheiros partilhados, para que as outras instancias saibam com que porta devam estabelecer a conexão.

Cada instancia deverá ter uma pasta chamada “shared”, e os ficheiros partilhados serão os que estiverem dentro desta pasta. Antes do envio da publicação, a aplicação deve verificar esta pasta atualizando a lista de ficheiros a partilhar.

Outra pasta necessária a cada instancia é a pasta chamada “download”, esta servirá para guardar o ficheiro descarregado. Caso já exista um ficheiro com o mesmo nome do ficheiro descarregado, então deverá ser substituído.

Protocolo UDP

O protocolo UDP é um protocolo muito simples que surgiu de forma a implementar algumas funcionalidades mínimas para uso direto de aplicações de rede de uso geral que o protocolo IP não implementa.

Este protocolo implementa, entre outras coisas, a deteção de erros nos dados e o mecanismo de identificação de aplicações.

Dentro de um nó existem números de porta que são utilizados para identificar aplicações individuais. Esta identificação é fundamental pois em cada nó podem existir centenas de aplicações de rede.

Os números de porta assumem um papel de elevada importância no modelo cliente-servidor, uma vez que estes servem, tal como referido anteriormente, para identificar aplicações. O cliente para além de precisar de conhecer o endereço IP da máquina onde está o servidor precisa de saber o número de porta que o servidor está a usar.

Exemplo: “192.200.10.6:200” – sendo que “192.200.10.6” se refere ao endereço do servidor que se encontra na porta 200. Por defeito a aplicação cliente assume que se pretende utilizar a porta “normal” para um dado serviço no caso de o utilizador não indicar ao cliente qual é o número de porta do servidor.

O protocolo UDP trata-se de um serviço de datagramas, ou seja, permite o envio de blocos de dados de tamanho variável que proporciona uma forma de identificação de aplicações individuais.

Sockets de rede em UDP

Um socket está associado a um número de porta local após o que fica em condições de receber e enviar datagramas UDP. Esta associação é condicionada pela obrigatoriedade de não existir já uma aplicação a usar o mesmo número de porta.

Na maioria das comunicações em rede é utilizado o modelo cliente-servidor, onde o servidor deve usar um número de porta fixo que é pré acordado com o cliente, contudo o cliente usa um qualquer número de porta (porta dinâmica).

Envio e receção de datagramas UDP

O serviço de datagramas UDP é não orientado à conexão e não oferece qualquer tipo de garantias. Cada datagrama é tratado de forma individual, logo para cada operação de envio de um datagrama

é necessário fornecer o endereço IP de destino e o número de porta de destino. Sob o ponto de vista da aplicação, a receção é normalmente síncrona, ou seja, a operação de receção bloqueia a execução (processo de thread) até que seja recebido um datagrama. Depois de ser recebido o datagrama UDP, a aplicação recetora tem acesso ao endereço IP de origem e número de porta da origem. Para o caso de se tratar de um servidor pode-se usar estes elementos para enviar uma resposta ao cliente posteriormente. O envio de datagramas UDP não oferece qualquer tipo de garantias, nem mesmo de feedback, ou seja, o emissor não tem forma de saber se o datagrama chegou ou não ao seu destino.

Envio e receção de datagramas

As operações de envio e receção de rede são geridas pelo sistema operativo através de filas FIFO (as primeiras operações a entrar são as primeiras a sair). Isto quer dizer que mesmo que a aplicação não solicite a receção de um datagrama, eles podem ser recebidos e serão disponibilizados à aplicação pela ordem em que chegaram, quando a aplicação solicitar a sua receção. O envio de datagramas é realizado através da invocação de um system-call que recebe como argumentos um bloco de dados com determinada dimensão e um endereço de destino (endereço IP + número de porta UDP). A system-call que solicita a receção de um datagrama devolve um bloco de dados com determinada dimensão e o endereço de origem do datagrama. Se não existir nenhum datagrama na fila a operação de receção bloqueia o processo/thread até que chegue um datagrama, logo, as receções de dados são bloqueantes.

Serviço não fiável – Como é um serviço muito simples, este protocolo não oferece nenhuma garantia, cabe às aplicações resolver os problemas que possam existir. As aplicações pouco cuidadosas podem facilmente ficar comprometidas por causa deste facto.

Tolerância a falhas

Cada pedido é tratado independentemente de pedidos anteriores. No caso de um modelo cliente-servidor, o servidor fica numa posição mais cómoda pois o seu funcionamento nunca será afetado se ocorrer falhas na entrega de datagramas. Contudo o cliente, depois de enviar o pedido, fica dependente da chegada de uma resposta do servidor. Para se resolver este problema deve-se evitar o bloqueio da aplicação quando fica à espera da receção da resposta definindo um tempo máximo para essa operação (timeout). Para tal existem algumas soluções:

- **Sockets não bloqueantes** - altera-se o comportamento de um socket de forma a que este se torne não-bloqueante; quando as operações que atuam sobre ele levarem a um bloqueio irá ser retornado de imediato um erro;
- **Sockets com timeout** - (ocorre apenas em algumas linguagens de programação como o Java) associa-se um tempo máximo para as operações que irão ser efetuadas ao socket; quando passar esse tempo é lançada uma exceção;
- **Threads** - o thread é criado para efetuar a receção sendo depois cancelado passado algum tempo (timeout) caso não tenha chegado nenhuma resposta;
- **Monitorização de sockets** - algumas API de sockets disponibilizam funções de monitorização; como exemplo temos o system-call select em C que permite monitorizar

um socket UDP para que este determine se existe ou não algum datagrama disponível para ser recebido. Esta função também permite associar um tempo limite à operação;

- **Sinais** - dependendo do sistema operativo, os processos são avisados da chegada de dados a um socket através de sinais (exemplo do sinal SIGIO nos sistemas UNIX).

Envio em broadcast

Broadcast é o processo pelo qual se envia informação para vários recetores ao mesmo tempo. Também podemos falar em *multicast* que é o processo de enviar informação apenas para um grupo de recetores e de *unicast* que pode ser considerado uma ligação ponto a ponto isto é um emissor envia a um recetor.

Ao contrário do serviço disponibilizado pelo TCP, o UDP permite o envio de dados para um endereço de broadcast.

Localização de aplicações - para efeitos de localização das aplicações usa-se UDP em broadcast utilizando umas das seguintes técnicas numa arquitetura cliente/servidor:

- **Anuncio de servidores** - um servidor envia periodicamente em broadcast um datagrama UDP para anunciar à rede a sua presença, dando, assim, a conhecer o seu endereço IPv4;
- **Pedido de servidores** - o cliente envia em broadcast um datagrama UDP para um número de porta pré combinado, onde solicita servidores. Estes últimos, por sua vez, respondem com endereços IPv4;

Em qualquer uma das técnicas anteriores os clientes criam uma lista de servidores disponíveis com respetivos endereços IPv4, sendo que depois podem escolher um e comunicar com ele usando UDP ou até mesmo TCP.

Tamanho dos datagramas

Um datagrama IPv4 pode ter 65535 bytes, logo o volume de dados num datagrama UDP poderia ter esse valor, descontando o tamanho do cabeçalho IPv4 (20 a 60 bytes) e o tamanho do cabeçalho UDP (8 bytes). Contudo, de forma a garantir que o volume excessivo de dados de um datagrama UDP não vai impedir a sua chegada ao destino devesse evitar ultrapassar os 512 bytes. Quando tal não acontece, ou se esquece os datagramas UDP e opta-se por conexões TCP ou então divide-se a informação por vários datagramas UDP.

Transações em múltiplos datagramas

Quando se opta por dividir a informação por vários datagramas é necessário se proceder a uma série de procedimentos de verificação. No mínimo, o recetor terá de saber o número total de datagramas e estes terão de ser numerados de forma a que não se perca informação e que esta depois possa ser reconstituída pela ordem correta. A falha de entrega de um datagrama pode levar a que toda a transação seja inútil o que levaria à sua repetição. O cliente, ao detetar uma falha de entrega de um datagrama pode solicitar ao servidor o envio apenas do datagrama em falta.

Associação de endereços remotos

É possível associar a um socket local um endereço remoto através do endereço IP e número de porta remotos, apesar do UDP ser um protocolo sem ligação. A associação de um socket UDP a um

endereço remoto tem efeito quer sob o ponto de vista de emissão de datagramas quer na sua receção:

- **Emissão** - cada datagrama deixa de necessitar de especificar o destino, sendo que o endereço remoto que foi associado ao socket é utilizado;
- **Receção** - o socket recebe apenas datagramas cujo endereço de origem corresponde ao endereço remoto que lhe foi associado.

Servidores UDP

Um servidor UDP tem normalmente um funcionamento muito simples devido às limitações existentes no protocolo UDP. O servidor limita-se apenas a receber um pedido no número de porta definido no protocolo de aplicação, sob a forma de um datagrama, processar o pedido e enviar a resposta ao cliente. Um servidor atende os pedidos por ordem de chegada, ou seja, um pedido que esteja na fila só é atendido depois de os pedidos à sua frente na fila terem sido processados e atendidos.

Protocolo TCP

Ao contrário do protocolo UDP, o TCP é muito mais complexo devido ao conjunto de funcionalidades disponibilizadas.

O TCP permite criar ligações lógicas bidirecionais entre aplicações residentes em nós de rede distintos. Estas ligações lógicas vulgarmente designadas conexões TCP fornecem garantias de entrega dos dados na ordem em que são emitidos.

As ligações TCP são exclusivas dos dois nós entre os quais são criadas, são canais de comunicação dedicados nos quais não é possível a intervenção de terceiros.

As transações de dados via TCP só são possíveis após o estabelecimento prévio da ligação TCP (conexão TCP).

Para ser possível criar uma conexão TCP as aplicações têm de desempenhar diferentes papéis:

- Uma das aplicações aceita o estabelecimento da ligação TCP (Servidor TCP);
- A outra aplicação pede o estabelecimento da ligação TCP (Cliente TCP).

Para que o servidor consiga desempenhar a sua função terá de associar um socket TCP a um número de porta fixo TCP previamente acordado com o emissor do pedido de ligação. De seguida fica à espera.

A aplicação cliente pode emitir um pedido de estabelecimento de ligação TCP especificando como destino o endereço IP do nó onde se encontra a primeira aplicação e o número de porta que essa aplicação está a usar.

O protocolo TCP tem a capacidade de entregar os dados de forma fiável, isto é, os dados são entregues pela ordem exata em que são emitidos.

Servidores TCP

Um servidor TCP é uma aplicação que aceita pedidos de ligação TCP num número de porta definido pelo protocolo de aplicação. Quando um servidor TCP aceita um pedido de ligação de um cliente, fica estabelecida uma ligação TCP e do lado servidor é criado um socket associado a essa ligação.

Servidores TCP multi-processo

Depois de um servidor TCP aceitar uma ligação de um cliente tem de se manter disponível para aceitar outros clientes, isto é, tem de comunicar com o cliente através do socket criado que corresponde à ligação TCP estabelecida entre eles, mas também tem de aceitar novos pedidos de ligação.

Resultados

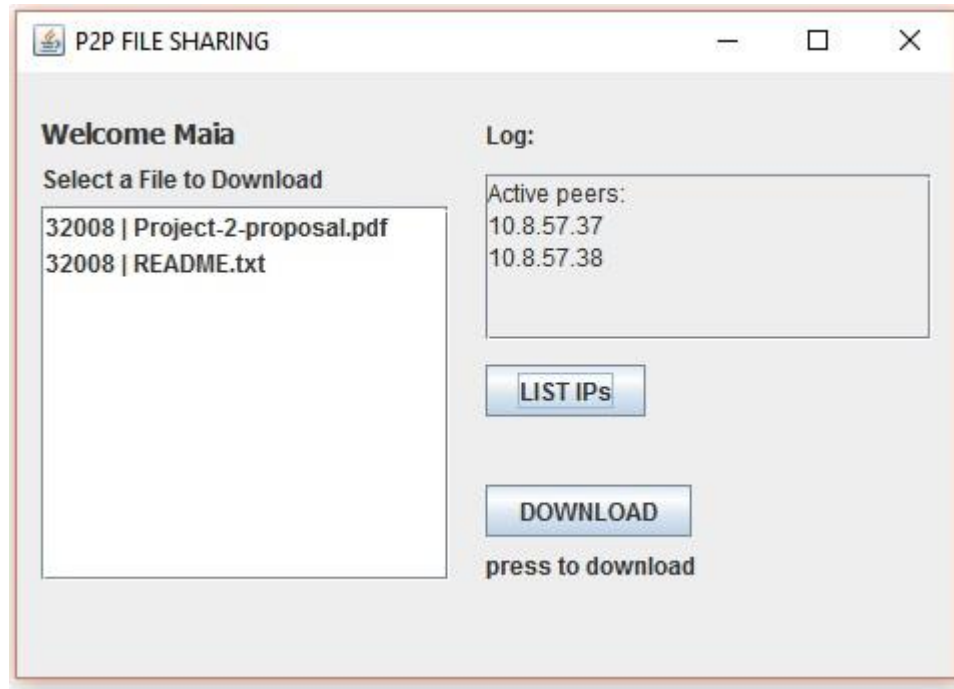


Figura 1- Main Window

A aplicação mostra uma lista, inicialmente vazia, de ficheiros disponíveis para download na rede, bem como uma janela de “Log” ou registo de interações com o utilizador.

A lista de ficheiros para download é atualizada a cada 45 segundos e é reescrita sempre que necessário adicionar ou remover um ficheiro da lista.

O “Log” por sua vez pode ser preenchido mecanicamente selecionando o botão “LIST IPs” que devolve a lista de IPs que estão a receber e a enviar “announcements” na rede através da aplicação.

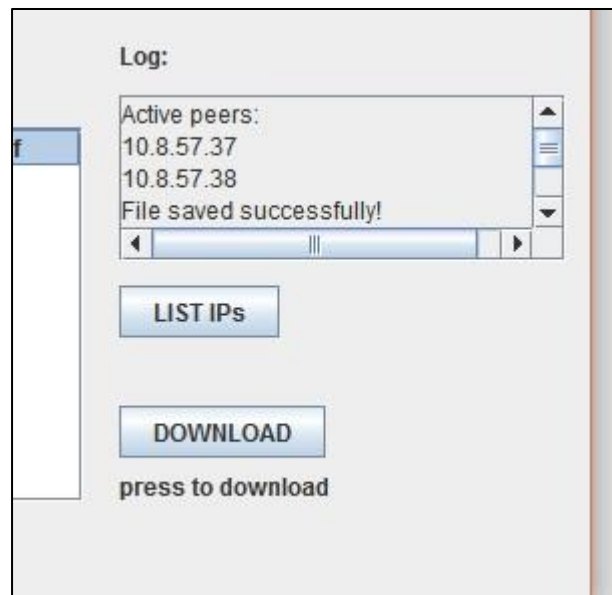


Figura 2- File Downloaded Successfully

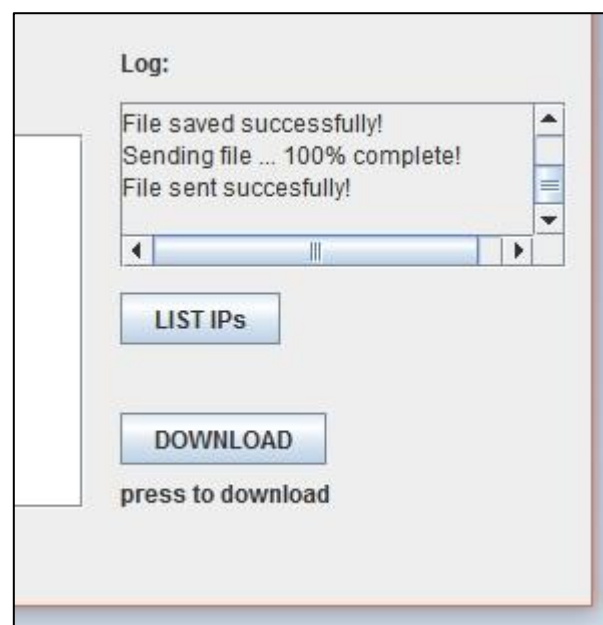
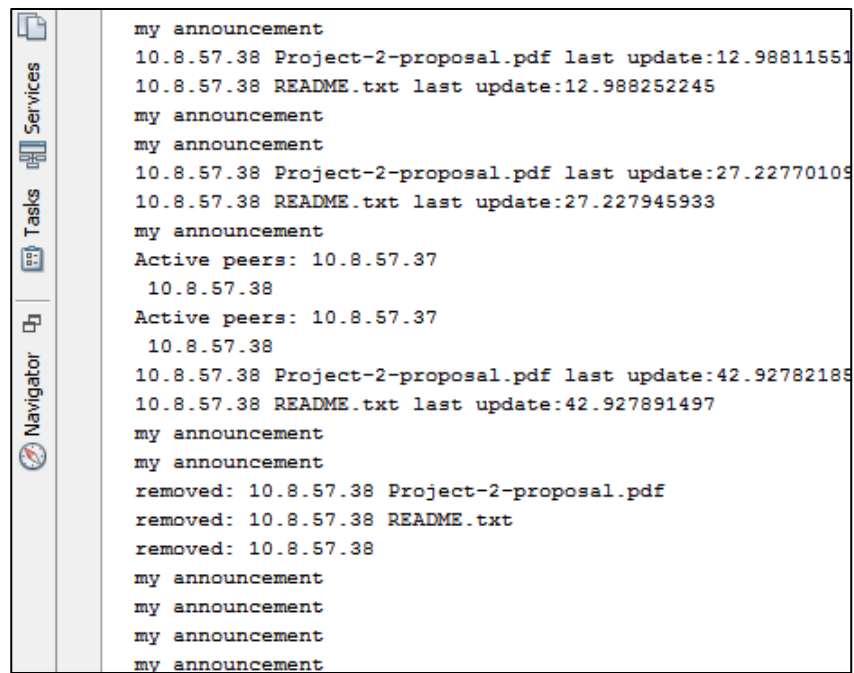


Figura 3 - Sending a file with success

O “Log” é ainda preenchido sempre que existe troca, ou tentativa de troca, de ficheiros entre utilizadores.

Sendo que sempre que um utilizador provoca uma interação, é notificado do sucesso da mesma, mas também quando envia um ficheiro é informado do seu sucesso e do seu estado de envio.



```

my announcement
10.8.57.38 Project-2-proposal.pdf last update:12.98811551
10.8.57.38 README.txt last update:12.988252245
my announcement
my announcement
10.8.57.38 Project-2-proposal.pdf last update:27.22770109
10.8.57.38 README.txt last update:27.227945933
my announcement
Active peers: 10.8.57.37
10.8.57.38
Active peers: 10.8.57.37
10.8.57.38
10.8.57.38 Project-2-proposal.pdf last update:42.92782189
10.8.57.38 README.txt last update:42.927891497
my announcement
my announcement
removed: 10.8.57.38 Project-2-proposal.pdf
removed: 10.8.57.38 README.txt
removed: 10.8.57.38
my announcement
my announcement
my announcement
my announcement

```

Figura 4 - background log

No registo de sistema é visível os *announcements* enviados bem como a receção de *announcements* de outros utilizadores. Sempre que um utilizador recebe informação, de um ficheiro de outro utilizador, é verificado se o mesmo já consta na lista e se sim então é atualizado o instante da última atualização. Assim quando um ficheiro não é atualizado por mais de 45 segundos o mesmo é removido da lista de ficheiros.

Análise de Resultados

Para a execução desta aplicação são utilizadas múltiplas *threads*. A “main thread” é onde corre o nosso ambiente gráfico e que faz o tratamento dos envios dos *announcements* em *broadcast*. Indui ainda métodos para a listagem dos peers ativos, novo download e saída do sistema. É nesta thread que existe o primeiro timer, que diz respeito aos announcements, o timer atualiza a cada 30 segundos.

```
fileTransferServer.start();
// Every 30s sends update from local files list
t = new Timer();
t.schedule(new TimerTask() {
    @Override
    public void run() {
        System.out.println("my announcement");
        try {
            P2PFileSharing.sendAnnouncement();
        } catch (InterruptedException | IOException ex) {
            Logger.getLogger(P2PFileSharing.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}, 0, 30000);

/*while (true) {*/
public static void exit() throws IOException, InterruptedException {
    t.cancel();
    UdpPeerReceive.t45.cancel();
    data[0] = 0;
    udpPacket.setData(data);
    udpPacket.setLength(1);
    for (i = 0; i < MAXCLI; i++) {
        if (peerActive[i]) {
            udpPacket.setAddress(peerAddress[i]);
            sock.send(udpPacket);
        }
    }
    ssock.close();
    sock.close();
    udpReceiver.join();
}
```

Figura 5 - Timer envio de announcement e método de saída

A “main thread” dá origem três novas threads, sendo a primeira de UdpPeerReceive. Nesta thread é tratada a receção de da informação proveniente do envio em broadcast sendo que se divide essencialmente em receção de novos peers, saída de um peer e announcements. É nesta thread também que é criado o timer responsável pela atualização da lista de ficheiros disponíveis para download. Este timer atualiza a cada 45 segundos e comunica diretamente com a lista visível para o utilizador.

```

p = new DatagramPacket(data, data.length);
t45.schedule(new TimerTask() {
    @Override
    public void run() {
        for (String key : mapa_Servidor_Ficheiros.keySet()) {
            for (FileInfo file : mapa_Servidor_Ficheiros.get(key)) {
                double lastupdate = (System.nanoTime() - file.getUptime()) / 1000000000.0;
                if (lastupdate > 45) {
                    mapa_Servidor_Ficheiros.get(key).remove(file);
                    System.out.println("removed: "+file.getEndereco_Servidor() + " " + file.getNome_Ficheiro());
                } else {
                    System.out.println(file.getEndereco_Servidor() + " " + file.getNome_Ficheiro() + " last update:" + lastupdate);
                }
            }
            if (mapa_Servidor_Ficheiros.get(key).isEmpty()) {
                mapa_Servidor_Ficheiros.remove(key);
                System.out.println("removed: "+key);
            }
        }
        P2PFileSharing.frame.updateList();
    }
}, 0, 45000);

```

Figura 6 - Timer de atualização da lista

A segunda thread criada logo após a anterior é a de FileTransferServer. Este servidor como o próprio nome indica gere o envio de ficheiros entre utilizadores através do protocolo TCP.

```

OutputStream os = cliSock[i].getOutputStream();

//Read File Contents into contents array
byte[] contents;
long fileLength = file.length();
long current = 0;

long start = System.nanoTime();
while (current != fileLength) {
    int size = 10000;
    if (fileLength - current >= size) {
        current += size;
    } else {
        size = (int) (fileLength - current);
        current = fileLength;
    }
    contents = new byte[size];
    bis.read(contents, 0, size);
    os.write(contents);
    P2PFileSharing.frame.addToLog("Sending file ... " + (current * 100) / fileLength + "% complete!\n");
    System.out.print("Sending file ... " + (current * 100) / fileLength + "% complete!");
}

os.flush();

P2PFileSharing.frame.addToLog("File sent successfully!\n");
System.out.println("File sent successfully!");
} catch (FileNotFoundException e) {
    P2PFileSharing.frame.addToLog(filename+"-File Not Found!\n");
}
}

```

Figura 7 - Parte do código do envio de um ficheiro

Esta thread comunica com a ultima thread criada que é a FileTransferClient. Este responsável pela receção de um ficheiro é a única thread que termina e é lançada varias vezes no correr de uma sessão da aplicação.

```
BufferedOutputStream bos = null;
InputStream is = socket.getInputStream();

//No of bytes read in one read() call
int bytesRead = 0;
int flag = 0;
while ((bytesRead = is.read(contents)) != -1) {
    if (flag == 0) {
        FileOutputStream fos = (created) ? new FileOutputStream(f.toString() + "/" + fileName) : new FileOutputStream(fileName);
        bos = new BufferedOutputStream(fos);
    }
    bos.write(contents, 0, bytesRead);
    flag = 1;
}
if (flag == 0) {
    P2PFileSharing.frame.addToLog(fileName + " Failed!\n");
} else {
    bos.flush();
    P2PFileSharing.frame.addToLog("File saved successfully!\n");
    System.out.println("File saved successfully!");
}
sOut.close();
socket.close();
```

Figura 8 - Receção de um ficheiro e fim da thread

Conclusão

Por fim, concluímos que o projeto foi concretizado com êxito e a aplicação cumpre os requisitos propostos.

Consideramos que este foi um trabalho que permitiu consolidar a matéria lecionada e aprofundar os conhecimentos adquiridos nas aulas.

A aplicação final é uma aplicação funcional e que julgamos que corresponde à intenção do trabalho, no entanto reconhecemos que alguns erros podem ser encontrados no correr da mesma.

Os problemas relativos a alguns nomes de ficheiros e o acesso remoto de múltiplos utilizadores a uma mesma porta não estão completamente resolvidos e podem causar falhas.

Referências Bibliográficas

ISEP. (s.d.). *Moodle ISEP*. Obtido em Maio de 2017, de Moodle ISEP
<https://moodle.isep.ipp.pt>

Glossário

- TCP - Transmission Control Protocol – protocolo que permite enviar e receber dados entre várias máquinas numa rede;
- UDP - User Datagram Protocol – protocolo que permite enviar e receber dados através de datagramas;
- WEB – rede que permite a conexão de vários computadores localizados em vários pontos do globo diferentes;
- Socket - Um socket de rede é o ponto-final de um fluxo de comunicação entre 2 aplicações através de uma rede;
- IP - Internet protocol – protocolo responsável pela identificação das máquinas e redes e pelo encaminhamento das mensagens até ao seu destino;
- Datagramas – unidade de transferência básica associada a uma rede de troca de dados.