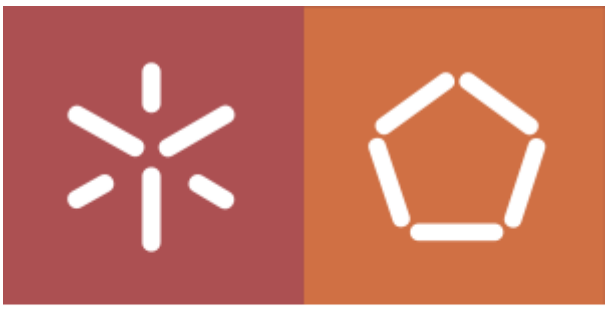

Universidade do Minho
Escola de Engenharia

Comunicações por Computador

Relatório TP2
Grupo 7

José Reis a87980
Mário Santos a70697
Nelson Ribeiro a93188

1. Introdução

No âmbito da Unidade Curricular de Comunicações por Computadores do 3º ano da Licenciatura em Engenharia Informática, foi-nos proposto a implementação de uma aplicação de sincronização rápida de pastas sem necessitar de servidores nem de conectividade Internet, designada por FolderFastSync (FFSync). Nesta aplicação, correm, em permanência, dois protocolos: um de monitorização simples em HTTP sobre TCP e outro desenvolvido pelo grupo, de raiz, para a sincronização de ficheiros sobre UDP.

Ao longo do documento será descrita a arquitetura da solução, todo o processo de especificação e implementação do protocolo, bem como a respetiva implementação e os resultados que demonstram as suas funcionalidades.

2. Arquitetura da solução

Com o objetivo de implementar uma aplicação de sincronização rápida de pastas sem necessitar de servidores nem de conectividade à Internet, a nossa arquitetura de solução tem três componentes principais: **FFsync**, **ClientHandler** e **PackBuilder**.

A classe *FFsync* contém a *main* da nossa aplicação, tendo como argumentos a *folder* e o endereço do parceiro com quem queremos sincronizar (*peer*). Uma vez validados os seus parâmetros, temos como objetivo estabelecer uma ligação, podendo começar de imediato a atender pedidos. Para tal, escolhemos a porta 8888 para maior comodidade e facilidade de testes.

O *ClientHandler* é a classe responsável por lidar com pedidos efetuados por outros peers. As comunicações são recebidas no *port* principal estabelecido, e posteriormente a informação é fornecida a uma instância de *ClientHandler,* este por sua vez trata de manipular e encaminhar a informação para o local devido.

O *PackBuilder*, como o nome assim o sugere, é um construtor dos diversos pacotes a enviar.

Existem ainda mais duas classes auxiliares na implementação da nossa aplicação: *FileInfo* e *LogBuilder*. A primeira calcula as diferenças entre duas listas de ficheiros através dos nomes dos ficheiros e da data de modificação, enquanto que a segunda monitoriza erros e determinadas métricas de desempenho, registando-os num ficheiro.

3. Especificação do protocolo

Formato das mensagens protocolares

Para assistir à criação do protocolo **FT-Rapid** criamos uma classe denominada **PackBuilder**, que vai servir como meio de armazenar a informação necessária a enviar em cada mensagem. Esta classe contém 5 variáveis que são usadas para definir as mensagens do protocolo FT-Rapid:

1) *pacote*: inteiro correspondente ao tipo do pacote;
2) *filename*: nome no ficheiro em questão;
3) *chunk*: inteiro correspondente ao número do bloco enviado em questão;
4) *tamanho_fich*: tamanho do ficheiro em questão;
5) *data*: informação que se quer enviar em bytes.

Para definir os diversos tipos de pacotes que serão enviados para permitir a sincronização de ficheiros entre peers, decidimos criar 6 tipos de pacotes, que serão identificados por um número inteiro correspondente, de 1 até 6. Os pacotes são os seguintes:

- Tipo 1 - Lista de nomes dos ficheiros que possui
- Tipo 2 - Lista de nomes dos ficheiros que precisa
- Tipo 3 - Transferência de ficheiros
- Tipo 4 - ACK
- Tipo 5 - FIN
- Tipo 6 - Erro

Tipo 1

As mensagens do tipo 1 contém uma Lista de Strings dos ficheiros que o peer que enviou esta mensagem possui, guardando essa informação no byte[] data. Para além da data, apenas necessita de identificar o pacote com o valor 1 e os restantes argumentos podemos ignorar, uma vez que são desnecessários.

Tipo 2

Neste tipo de mensagem podemos ignorar o conteúdo das variáveis *filename, chunk* e *tamanho_fich*, uma vez que não é necessário passar este tipo de informação, apenas precisamos de enviar na *data* uma Lista de Strings com os dos ficheiros que o peer que enviou esta mensagem precisa.

Tipo 3

Aqui é onde a transferência de ficheiros ocorre, por isso, todas as variáveis vão ter de ser preenchidas. Precisamos de identificar a que ficheiro pertence a chunk que estamos a transmitir, qual é a posição total desta chunk, ou seja, em que

posição esta chunk encontra-se quando juntarmos todas as chunks, o tamanho total do ficheiro e na *data* temos o chunk do ficheiro em bytes.

Tipo 4

Com este tipo de mensagem queremos avisar ao *peer* que nos está a transmitir um ficheiro quais os chunks que já recebemos. Deste modo, vamos preencher a variável *data* com uma Lista de inteiros que correspondem aos *chunks* que já recebemos. Com esta informação o outro peer vai averiguar quais chunks se perderam e enviá-los de novo.

Tipo 5

O objetivo desta mensagem é avisar que a transferência de ficheiros já terminou. Portanto, apenas é necessário identificar o tipo do pacote, que o sistema ao receber este tipo de mensagem já vai saber o que necessita de fazer sem informação extra.

Tipo 6

Nesta mensagem vamos preencher a *data* com uma String que corresponde ao nome do erro que ocorreu.

Para uma melhor facilidade de compreensão e análise do protocolo em questão, iremos exemplificar como decorreria a sincronização de ficheiros de apenas um peer.

Primeiramente, é enviada a sua lista de ficheiros (TIPO 1) de forma a mostrar ao outro *peer* os ficheiros que pode enviar. De seguida, após a análise da lista fornecida, o outro peer devolve uma mensagem com a lista de ficheiros que necessita (TIPO 2).

Uma vez obtida a lista dos ficheiros que precisa sincronizar, vai iterar essa lista. Para cada ficheiro vai-se dividir o ficheiro em chunks de 1024 bytes, para de seguida os enviar sequencialmente para o outro *peer*. Quando acabar de enviar os chunks todos de um ficheiro vai enviar uma mensagem a pedir ao outro *peer* para confirmar os chunks que recebeu. Se o outro peer já tiver recebido todos os chunks necessários vai começar a enviar os chunks respetivos ao próximo ficheiro, ou, caso não haja mais ficheiros, vai enviar uma mensagem para finalizar a conexão (TIPO 5). Acabando desta maneira sincronização.

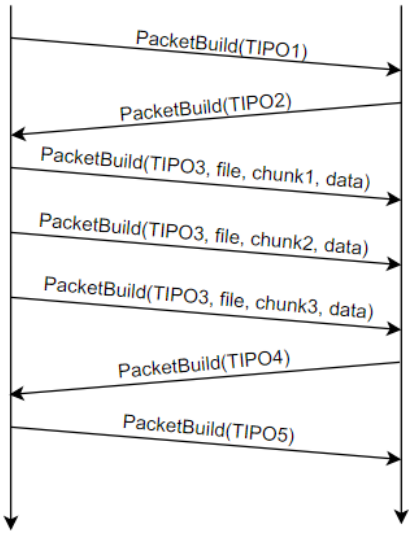

Figura 1- Diagrama temporal do protocolo FT-Rapid

4. Implementação

No momento inicial de execução *FFsync* lança duas Threads fundamentais: uma efetua *request* de sincronização a um peer, outra recebe pedidos de peers e responde a esses pedidos. Ambas se encontram à escuta na porta 8888.

Ao ser inicializada, a primeira thread envia a lista de ficheiros existentes no folder que pretendemos sincronizar, contudo, como esta é a primeira a iniciar e ainda não se encontra a outra thread ativa do outro lado para receber esta mensagem, optamos por enviar novamente quando este peer receber a lista de ficheiros de outro parceiro. Após recepção da lista de ficheiros do parceiro calculamos os elementos em falta de sincronização com recurso ao método *NeededToSend* e enviamos as suas referências num packet do tipo 2 através do socket correspondente. O peer, ao receber o pacote enviado, fará uso de uma instância da classe *ClientHandler* para identificar os ficheiros a enviar, convertê-los para bytes, dividi-los em chunks ordenados e efetuar o envio através de packets tipo 3 (*file data*). No final da sincronização são enviados packets tipo 4 (confirmação) com informação de que a comunicação foi efetuada. Após o recebimento desta informação é transmitido um *packet* do tipo 5 com informação de que a conexão deve ser terminada.

5. Testes e resultados

Para os testes e resultados, foi utilizada a topologia do core fornecida no início do ano. Como teste ao nosso sistema corremos o FFSync em dois nós diferentes, neste caso no Servidor1 e no Golfinho, e como podemos averiguar na figura abaixo a transferência foi feita e no fim obtemos os mesmos ficheiros nas duas diretorias diferentes.

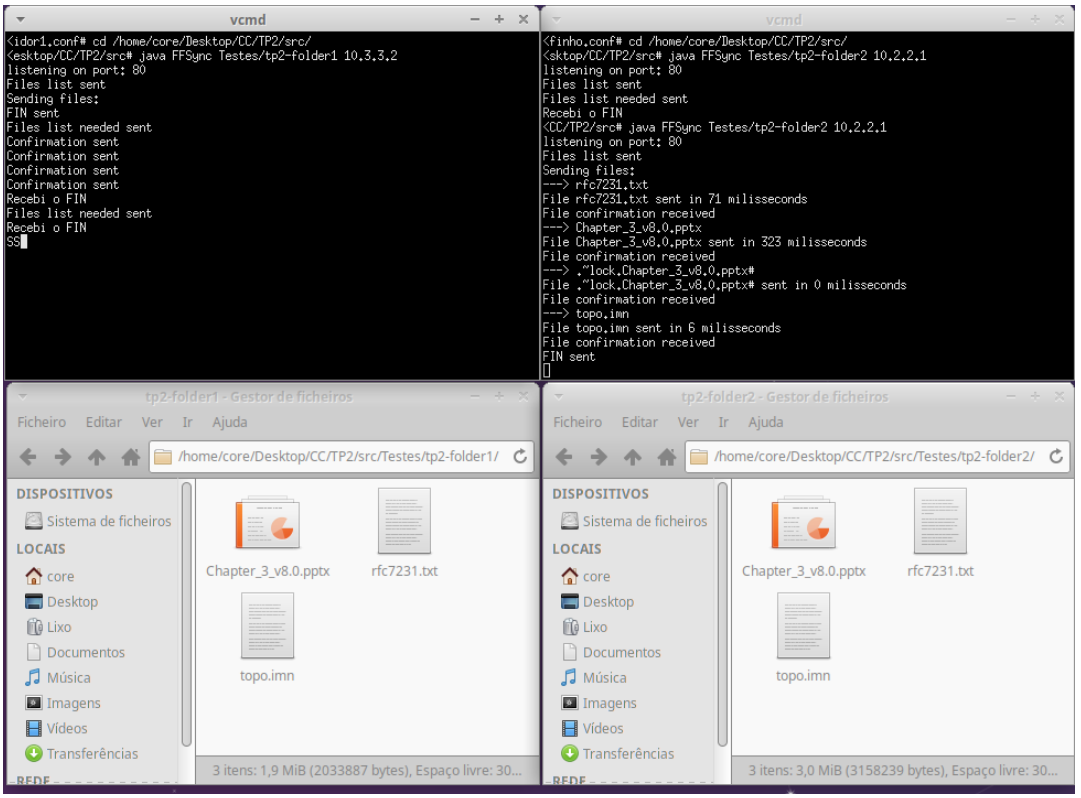

Figura 2: Teste 1

Para garantir que os ficheiros foram bem transferidos utilizou-se o comando *diff* entre as duas diretorias e como podemos verificar os resultados foram positivos.

```
core@core-VirtualBox:~/Desktop/CC/TP2/src/Testes$ diff tp2-folder1/rfc7231.txt t
p2-folder2/rfc7231.txt
core@core-VirtualBox:~/Desktop/CC/TP2/src/Testes$ diff tp2-folder1/topo.imn tp2-
folder2/topo.imn
```

Figura 3: Utilização do comando diff para os ficheiros das diferentes diretorias

6. Conclusões e trabalho futuro

Neste trabalho ainda existem várias adições e otimizações que poderiam e deveriam ser implementadas no protocolo desenvolvido. Isto por questões de tempo e um arranque “lento” na realização deste projeto, uma vez que demoramos a adquirir os conhecimentos necessários e a estruturar o protocolo em questão.

Um ponto crucial no projeto que não foi possível desenvolver foi a transferência de ficheiros por concorrência, era apenas necessário criar uma thread para cada ficheiro que seria enviado. Não seria difícil de implementar no código existente, uma vez que já se encontrava bem estruturado de tal forma, contudo deparamo-nos com alguns *bugs* de complexidade considerada.

Reparamos que depois de definir os diversos tipos de pacotes, em parte deles existe informação redundante e não necessária para o pacote em questão, pelo que seria uma otimização natural de se fazer.

Este trabalho foi muito importante para o nosso aprofundamento deste tema, uma vez que nos permitiu ficar a conhecer melhor a estrutura basilar dos protocolos que usamos diariamente nas nossas comunicações.