Folder Fast Sync
Sistema de Sincronização de Pastas

António Fonseca[1][93167], Diogo Rebelo[2][93278] e Henrique Alvelos[3][93316]

[1] Universidade do Minho, Gualtar, Braga, Portugal
a93167@alunos.uminho.pt
[2] Universidade do Minho, Gualtar, Braga, Portugal
a93278@alunos.uminho.pt
[3] Universidade do Minho, Gualtar, Braga, Portugal
a93316@alunos.uminho.pt

Abstract. Given the current need to work with the constant use of cloud storage services, this project is primarily related to the construction of an application capable of ensuring the need to keep several folders synchronized (both with the same files), in real time. This basic functionality with regard to the root of teamwork, must be ultra fast and implemented using the UDP protocol, which is not connection oriented and inevitably faster.

Keywords: Computer Network · Protocols · Transport Layer · Sockets · Java programming.

Resumo. Perante a necessidade atual de trabalhar com o recurso constante a serviços de armazenamento na nuvem, surge este projeto, que se prende primordialmente com a construção de uma aplicação capaz de ver assegurada a necessidade de manter várias pastas sincronizadas (com as mesmos ficheiros), em tempo real. Esta funcionalidade básica no que diz respeito à raiz do trabalho em equipa, deve ser ultra rápida e implementada utilizando o protocolo UDP, não orientado à conexão e inevitavelmente mais rápido.

Palavras-Chave: Redes de Computadores · Protocolos · Camada de Transporte · Sockets · Programação em Java.

1 Introdução

1.1 Contextualização

O presente relatório tenta explorar de uma forma mais profunda alguns conceitos primordiais sobre o funcionamento prático dos protocolos de transporte (TCP UDP), muito bem estudados no trabalho prático 1. Este projeto desenvolvido em JAVA prende-se muito com a implementação de uma aplicação de sincronização rápida de pastas que não se socorra de conectividade à Internet ou de servidores. Assim sendo, ao longo do trabalho referir-nos-emos à aplicação com a abreviatura FFSync, isto é, *Folder Fast Synchronization*.
No final do trabalho, estarão concretizados com sucesso os seguintes objetivos:

- Implementar um protocolo próprio que saiba gerir as principais funcionalidades da aplicação;
- Construir um servidor simples em HTTP, capaz de monitorizar o estado do sistema;
- Compreender o modo de funcionamento dos diferentes protocolos existentes;
- Desenvolver um maior conhecimento em relação à utilização de Sockets na linguagem de programação utilizada.

1.2 Modo de Utilização

Ao iniciar a aplicação, devem ser indicados alguns parâmetros indispensáveis à sincronização, nomeadamente, a pasta a sincronizar e o endereço de IP do respetivo parceiro. Estes parâmetros são aceites via terminal no formato $ FFSync folderName peerIPaddress. Assim que as condições mínimas estejam reunidas (formato válido dos dados e acesso à rede), são tratados de imediato pedidos TCP e, simultaneamente, pedidos UDP, nas respetivas portas para o efeito. Em relação aos pedidos UDP, visa-se a criação de um servidor HTTP simples capaz de, perante pedido do utilizador, apresentar o estado de determinado pedido (via browser).

2 Arquitetura da Solução

Em relação à solução apresentada, o projeto divide-se em 4 packages essenciais que visam manter mais organização. Cada package surge descrito a seguir, assim como as respetivas classes.

- **Connect:** Aglomera o conjunto de classes responsáveis por iniciar, manter e terminar a conexão entre os *peers*. Mais especificamente, contém as classes dos diferentes servidores TCP e UDP;
- **LogFile:** Aglomera o conjunto de classes responsáveis por gerar o ficheiro de Log e mantê-lo atualizado com a informação requesitada;
- **Manager:** Aglomera o conjunto de classes responsáveis por gerir uma lista de pacotes e/ou de ficheiros;
- **Packet:** Aglomera o conjunto de classes responsáveis pela criação de cada pacote. Auxilia de modo mais íntimo o protocolo desenhado.

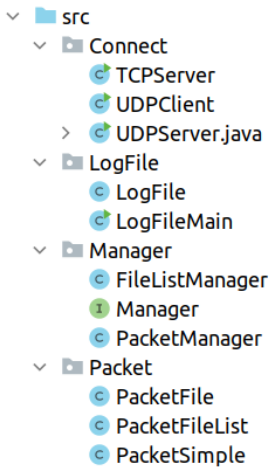

Fig. 1: Arquitetura da aplicação.

3 Especificação do Protocolo

Esta constituiu a primeira fase de trabalho, mostrando-se extremamente relevante para todo o desenvolvimento do trabalho, já que é em torno deste protocolo que a sincronização acontece. Esta especificação divide-se em três partes, nomeadamente, a sintaxe, a semântica e o comportamento do protocolo edificado.

3.1 Sintaxe

No desenho do protocolo FT-Rapid solicitado, surgem vários campos que urgem ser apresentados. Neste contexto, o formato das mensagens protocolares criado permite que a transferência de informação entre clientes aconteça, seguindo inclusive essa mesma semântica. Com efeito, segue-se essa mesma constituição. Primeiramente, surge a constituição de um pacote de ficheiros e, depois, de uma lista de ficheiros.

Olhando para os modelos abaixo, extrai-se a seguinte informação:

- Surge reunida a informação alusiva a um pacote de ficheiros e como lidar com ele, seja em termos de ACK's, seja em termos de erros;
- Relativamente aos ACK's, existem diferentes formatos de pacotes, consoante a situação: tudo recebido do ficheiro, apenas um pacote recebido de um ficheiro ou tudo enviado.
- Relativamente aos Erros, procede-se de igual modo, há um formato para erro na lista de ficheiros e outro para forçar novamente o envio.

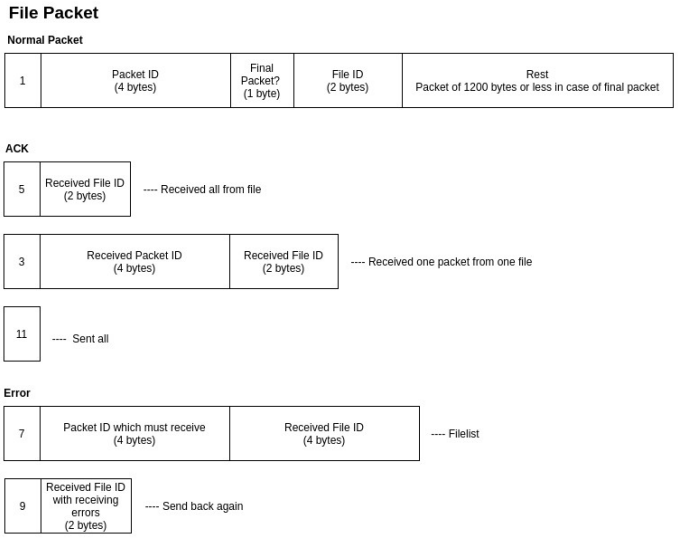

Fig. 2: Formato de um Pacote de ficheiros.

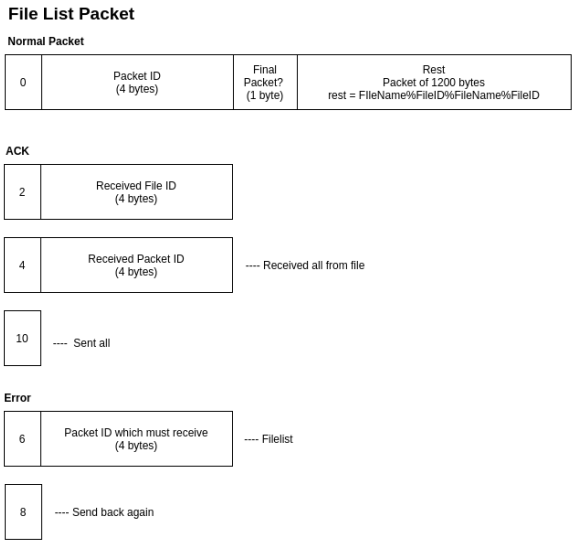

Fig. 3: Formato de um Pacote de ficheiros.

3.2 Semântica

O formato do protocolo apresentado especifica diferentes pacotes e com diferentes campos. É, então, necessária a sua explicação.

File Packet: Explicação dos diferentes campos. O número de bytes ocupado para cada campo surge no modelo.

1. **Normal Packet:** Começa por ter um primeiro campo que serve para diferenciar os diferentes pacotes. Segue-se o ID do pacote e o próximo campo serve para se saber se o pacote em questão é o último. De seguida, temos o ID do ficheiro em questão, seguindo-se o resto do pacote nos 1200 bytes seguintes (caso o primeiro byte seja 1, tem-se 1200 bytes do ficheiro; caso seja 0, tem-se 1200 bytes do conteúdo da lista de ficheiros).
2. **ACK:** Temos diferentes casos. Caso tudo tenha sido recebido do ficheiro: o pacote começa por ter um primeiro campo utilizado para diferenciar os diferentes pacotes. Segue-se o ID do ficheiro recebido. Caso se tenha recebido um só pacote de um ficheiro: o pacote começa por ter o seu índice, seguindo-se o ID do pacote recebido e o ID do ficheiro recebido. Caso tudo tenha sido enviado, o pacote tem apenas o índice.
3. **Error:** Distinguem-se dois casos: Se o erro se deu com a lista de ficheiros, o pacote tem o seu índice, o ID do pacote a receber e o ID do ficheiro recebido. Se é necessário enviar novamente o ficheiro, o pacote tem o seu índice e o ID do ficheiro com erros na receção.

File List Packet: Explicação dos diferentes campos. O número de bytes ocupado para cada campo surge no modelo.

1. **Normal Packet:** Começa por ter um primeiro campo que serve para diferenciar os diferentes pacotes. Segue-se o ID do pacote e o próximo campo serve para se saber se o pacote em questão é o último. De seguida, temos o resto do pacote nos 1200 bytes seguintes.
2. **ACK:** Temos diferentes casos: o primeiro, com o índice e o ID do ficheiro recebido e, o segundo, com o índice e o ID do pacote recebido. Caso tudo tenha sido enviado, o pacote tem apenas o índice.
3. **Error:** Distinguem-se dois casos: Se o erro se deu com a lista de ficheiros, o pacote tem o seu índice e o ID do pacote a receber. Se é necessário enviar novamente o ficheiro, o pacote tem apenas o seu índice.

3.3 Comportamento

Para descrever o comportamento do protocolo e como este atua, nada melhor do que utilizar diagramas temporais que elucidem o respetivo fluxo. Sendo assim, surgem os diferentes diagramas, estando na legenda de cada figura a respetiva situação. Contudo, cada pacote é tratado por Threads, sendo que cada thread representa um ID de um ficheiro.

Relativamente à File List

Caso 1: Envio do pacote com Index igual a 0 e recebido Ack com Index igual a 2.

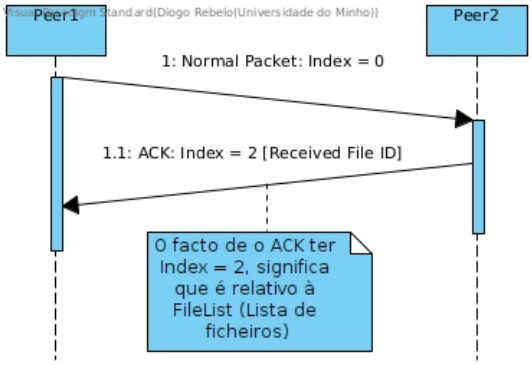

Fig. 4: Diagrama Temporal alusivo ao Caso 1.

Caso 2: Envio do pacote com Index igual a 0 e recebido Ack com Index igual a 4.

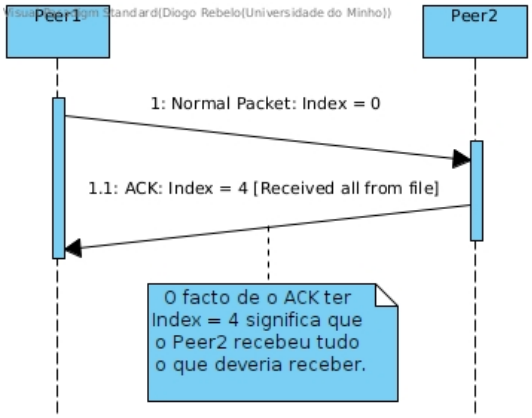

Fig. 5: Diagrama Temporal alusivo ao Caso 2.

Caso 3: Envio do pacote com Index igual a 0 e recebido Ack com Index igual a 6.

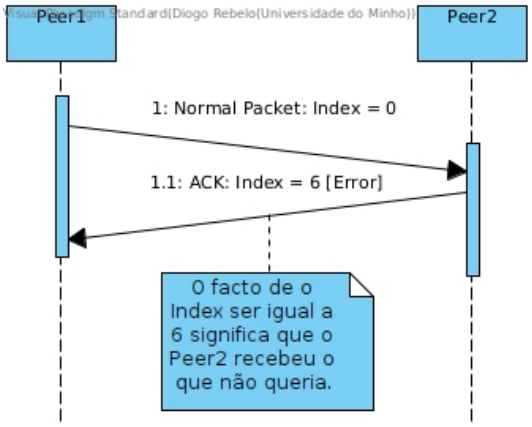

Fig. 6: Diagrama Temporal alusivo ao Caso 3.

Relativamente a um File

Caso 1: Envio do pacote com Index igual a 1 e recebido Ack com Index igual a 3.

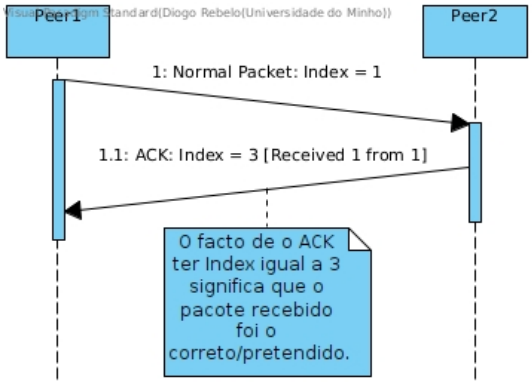

Fig. 7: Diagrama Temporal alusivo ao Caso 1.

Caso 2: Envio do pacote com Index igual a 1 e recebido Ack com Index igual a 5.

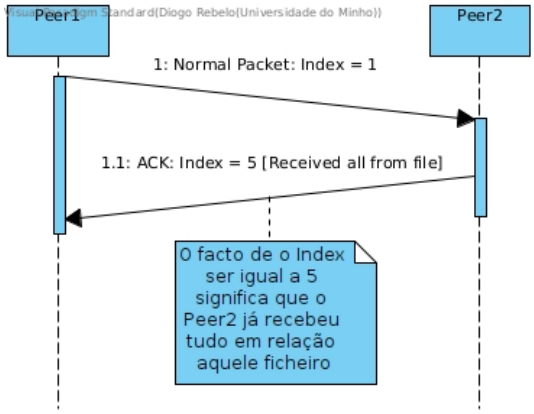

Fig. 8: Diagrama Temporal alusivo ao Caso 2.

Caso 3: Envio do pacote com Index igual a 1 e recebido Ack com Index igual a 7.

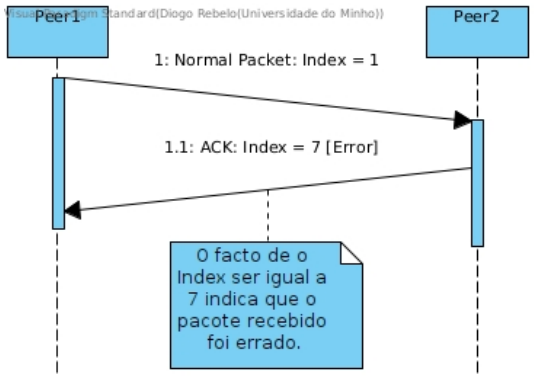

Fig. 9: Diagrama Temporal alusivo ao Caso 3.

Fluxo Comportamental: Segue-se o fluxo de atividade idealizado, para melhor perceção.

1. Servidor fica a espera que algo aconteça;
2. Cliente manda um Datagram Packet;
3. Servidor recebe e cria um clientHandler para enviar a FileList;
4. Ao mesmo tempo, o cliente cria um ClientHandler para enviar a Filelist, recebendo pacotes da lista de ficheiros;
5. Tanto um como outro mandam ACK's para dizer que receberam x pacote (acabando de receber a lista de ficheiros do outro, fazem a diferença de ficheiros);
6. Começam a criar Threads para cada ID de ficheiro;
7. Enviam pacotes de vários ficheiros, terminando isso, acaba a conexão.

4 Implementação

A aplicação desenvolvida teve como linguagem de programação o JAVA e socorreu-se principalmente de bibliotecas de Sockets para o efeito. Nada melhor do que o diagrama de classes da aplicação desenvolvida para nos aproximar com a implementação. A implementação foi testada e está funcional na IDE IntelliJ.

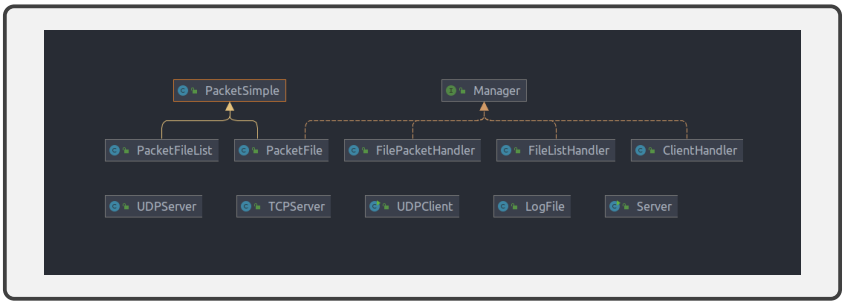

Fig. 10: Diagrama de Classes simples da Aplicação.

5 Conclusão e Trabalho Futuro

Naturalmente, o trabalho realizado contribuiu para uma melhor compreensão da matéria lecionada nas aulas teóricas e práticas de um modo geral. O mais relevante em relação ao projeto foi compreender a complexidade crescente dos assuntos tratados: construir por palavras o que pretendíamos fazer era uma tarefa razoavelmente simples, todavia, transpor tudo isso para a realidade prática com código propriamente dito, já não se mostrou da mesma dificuldade.

O grupo começou por desenhar o protocolo, quanto à sua sintaxe, semântica e comportamento, como proposto pelo docente. Com esta edificação, passamos para a implementação propriamente dita. Mesmo assim, é de frisar que à medida que a implementação foi sendo feita, o protocolo foi aprimorado e repensado. Em relação a esta fase, socorremo-nos do Java e das suas bibliotecas de Sockets indispensáveis na resolução. O servidor básico em HTTP e a gestão de Logs em ficheiro foram assuntos mais tardios e que, posteriormente, seriam interligados com o restante projeto.

Ao longo do desenvolvimento, foram imensas as dificuldades que surgiram, nomeadamente como permitir que vários peers se ligassem para a respetiva troca, como efetuar a transferência de bytes dos ficheiros, que aspetos teria o protocolo de ter em conta, como organizar toda a informação em jogo. Quanto ao trabalho realizado, o grupo considera que vê concretizados os requesitos mínimos, contudo, naturalmente, existem aspetos a ser melhorados, com o auxílio dos testes, não realizados, devido ao tempo.

6 Sobre os autores

António Fonseca

Diogo Rebelo

Henrique Alvelos