TP2 Serviço de transferência rápida e fiável de dados sobre UDP

Alexandre Martins, João Bernardo Freitas, and João Luís Amorim

University of Minho, Department of Informatics, 4710-057 Braga, Portugal
e-mail: {a77523,a74814,a74806}@alunos.uminho.pt

1 Introdução

Neste projeto era necessário desenvolver um Serviço de transferência rápida e fiável de dados tendo como base o UDP. A transferência deveria funcionar para ambos os lados, tanto upload como download. Deveriamos assegurar que o nosso serviço teria um controlo de conexão, assim como um mecanismo de verificação de integridade (*Checksum*) e outro capaz de confirmar a transmissão (ACK) ou retransmissão caso seja necessário, para controlar os erros que pudessem ocorrer.

A partir daqui desenvolvemos uma arquitetura para o nosso sistema tendo como base um Protocol Data Unit (PDU) também desenvolvido por nós.

2 Especificação do Protocolo

Para dar início à apresentação do trabalho desenvolvido, vamos começar por apresentar definição de PDU criada que será enviada entre AgentesUDP.

2.1 PDU:

Após várias mudanças, decidimos que para garantir uma tranferência fiável de dados o nosso *PDU* teria de ter um tipo, indicando este o tipo de pacote que estamos a transferir. Este *PDU* pode ser um *PUT* ou *GET* que corresponde a upload e download de ficheiros, respetivamente, um *ACK*, para efeitos de confirmação do recebimento dos pacotes e retransmissão daqueles que foram perdidos, pode ainda ser do tipo *CHECKSUM* que nos permite saber se houve algum erro durante a transferência, funcionando como mecanismo de verificação de integridade, marcando também o final da transferência dos ficheiros, e por fim, pode ser do tipo *DATA*, que como o nome indica serve para transportar os dados do ficheiro a transferir. De referir que em cada PDU temos o primeiro byte para indicar o tipo, os 4 seguintes para indicar o tamanho do pacote a enviar, os próximos 4 para indicar o offset e se o PDU for do tipo Data, os restantes 512 bytes para o envio dos dados.
Apresentamos de seguida, a definição em Java do PDU:

```
public class PDU implements Serializable{
    //Declaração dos tipos de PDU
    public static final int TYPE_ACK = 1;
    public static final int TYPE_PUTFILE = 2;
    public static final int TYPE_DATA = 3;
    public static final int TYPE_CHECKSUM = 4;
    public static final int TYPE_GETFILE = 5;

    //Variável correspondente ao tipo de PDU
    private int type;
    //Variável correspondente ao tamanho do PDU
    private int length;
    //Variável correspondente ao offset do PDU
    private int offset;
    //Variável correspondente aos dados enviados no PDU
    private byte[] data;
```

Figura 1: Definição de PDU.conf

2.2 Interações:

Tal como era pedido no enunciado, é necessário que exista uma entidade que trata do recebimento e envio dos pacotes, tendo este de enviar posteriormente estes pacotes para uma camda que seria responsável pelo tratamento dos pacotes recebidos. No caso da nossa aplicação, o AgenteUDP é o responsável pelo envio/recebimento dos pacotes, enviando-os então para a camada TransfereCC onde estão implementadas as funcionalidades necessárias para que a transferência ocorra como pretendido. De notar que incialmente foi criado algum tratamento dos PDU's, tanto para enviar como para receber, no AgenteUDP. Apesar de o nosso grupo saber que qualquer tipo de tratamento de informação deveria ser tratado na camada TransfereCC, ainda precisavamos de implementar a verificação de integridade e o término da conexão no pouco tempo que tinhamos e por isso mesmo resolvemos não alterar uma vez que aquilo que é feito consiste apenas na verificação do tipo de PDU's que estão a ser recebidos e enviados e na sua conversão para bytes ou inteiros.

É importante ter em atenção que ao longo da execução da transferência dos ficheiros, a camada TransfereCC comunica com uma espécie de tabela onde é guardado o estado da transferência a cada momento, sendo este ponto fulcral para que toda a transferência ocorra sem erros. No final, a Main, que corresponde à CmdLineApp mencionada no enunciado, trata de receber os argumentos necessários para dar inicio à transferencia dos ficheiros, ao criar uma instância de AgenteUDP e outra de TransfereCC, em cada ponta da transferência.

Na próxima secção será feita uma apresentação a todos os módulos criados que não correspondem ao PDU, uma vez que este já foi apresentado.

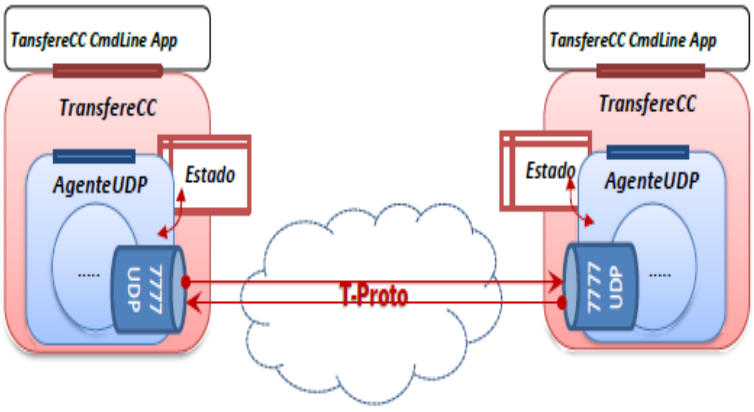

Figura 2: Transferência de Ficheiros sobre UDP.conf

3 Implementação

3.1 Módulos

Uma vez que a definição e constituição do PDU já foi apresentada, passemos para a apresentação dos outros módulos que foram implementadas para a construção de toda a aplicação.

AgenteUDP: Neste módulo assenta toda a comunicação, sendo por este recebidos e enviados todos os datagramas *UDP* . Quando é recebido um pacote, o agenteUDP cria um *PDU* com o tipo de pacote recebido, com o tamanho, o offset e os dados do pacote e envia posteriormente este *PDU* para a outra camada denominada por *TransefereCC*, onde foram implementadas as funcionalidades necesárias para que a transferência ocorra como pretendido. Por outro lado, quando queremos enviar um pacote, este envio será feito na mesma pelo *AgenteUDP* e o tratamento dos pacotes a enviar será igualmente efetuado pela camada *TransefereCC*. De notar que aqui é também definido o tamanho máximo de dados enviados em cada pacote, sendo este igual a 512 bytes.

Quando queremos enviar informação, cria um *PDU* e passa-o para outra camada, *TransfereCC*. O contrário acontece quando recebemos informação. É-lhe passado um pacote da camada *TransefereCC* que de seguida é tratado de modo a retirar a informação pretendida.

TransfereCC: Na camada TransfereCC, era necessário implementar as funcionalidades necessárias para que a transferência de ficheiros ocorresse sem qualquer falha. Aqui é importante realçar que o "core"da camada para o envio dos ficheiros está presente no método sendFile(), enquanto que para o recebimento de ficheiros está no método receive(). A ideia base desta camada consistiu na criação inicial do envio de ficheiros com base no comando "put", isto porque, uma vez implementado o comando "put"e enviado o ficheiro, se recebessemos um comando "get", apenas teríamos que enviar um PDU a indicar este mesmo tipo "get"e de seguida tratar o envio do ficheiro pedido como um comando do tipo "put". Para o recebimento de ficheiros, a ideia inicial passou por, em primeiro lugar, determinar que tipo de PDU's estavamos de facto a receber, tal como é implementado no método receive(). De seguida, aquilo que seria preciso seria interpretar e tratar cada tipo de PDU, com o auxilio de métodos como o *handleACK* de modo a que conseguissemos utiliza-los de forma a dar continuidade à transferência dos ficheiros.

Estado: Para controlar e manipular a transferência de dados de um sítio para outro foi necessária a criação de uma classe Estado, que contêm informações como o estado da transferência, o nome do ficheiro, o número de pacotes a enviar, o tamanho do ficheiro a enviar, o tamanho de cada pacote a enviar, o comando inserido e a indicação da conclusão da transferência do ficheiro. De notar que o estado da transferência vai sendo atualizado ao longo da execução da aplicação, permitindo que haja uma distinção entre o que está de facto a ocorrer na execução do programa, como por exemplo, se já recebemos um ACK correspondente à confirmação de um pacote enviado ou não. Sem a existência desta classe não nos seria possível distinguir as várias etapas do envio e recebimento de cada pacote.

EstadoPacket: Esta classe foi criada com o intuíto de ter um estado para cada pacote transmitido, onde guardamos a informação relativa ao tempo a que foi transmitido e qual o offset do pacote, no caso do ficheiro ser fragmentado. Estas informações

serão usadas posteriormente para, por exemplo, a retransmissão de pacotes, em que será criada uma coleção de Estados de Pacote, correspondente ao conjunto de pacotes enviados. Uma vez recebidos os ACK's para estes pacotes enviados, estes serão removidos da coleção. No final apenas serão retransmitidos aqueles que ainda permanecerem dentro desta coleção.

Main: A main funciona como aquilo que é referido no enunciado como CmdLineApp. Aqui é feito o tratamento dos argumentos que serão recebidos aquando da execução do programa, em que o primeiro corresponderá a um "wait", "get"ou "put". No caso do "wait", o segundo argumento corresponderá à porta local. No caso do "get"ou "put", o segundo argumento corresponderá ao endereço para onde se pretende enviar o ficheiro, o terceiro à porta local e o quarto à porta remota. De notar que a introdução dos comandos é "case sensitive", sendo apenas aceites comandos com letra minuscula.

Como referido anteriormente, o sistema deveria ser capaz de fazer um controlo de conexão, um controlo de erros e uma verificação de integridade dos dados.

Controlo de conexão: O controlo da conexão é feito a partir dos tipos de pacote *PUT* e *GET*. Ao ser enviado tanto um como o outro, é esperada uma resposta em forma de *ACK* para confirmar que a ligação foi estabelecida com sucesso e prosseguir com a transferência. Término da ligação? TODO

Controlo de erros: Para confrontar o problema dos erros na transmissão, implementamos no nosso projeto um sistema de *acknowledgments*, *ACKs*, que nos permite confirmar se o pacote foi ou não recebido pelo destinatário. Dentro dos *ACKs* temos a distinção entre *ACKs* de confirmação a um comando "put"ou "get, de erro a um comando "get"e obviamente a um *ACK* de confirmação de recebimento do pacote de dados.

ACK

Verificação de Integridade: Para verificação de integridade foi utilizado o algoritmo MD5 ("Message-Digest algorithm 5"), que corresponde a uma função de dispersão criptográfica (ou função hash criptográfica) de 128 bits unidirecional. O funcionamento que foi implementado consiste, do lado de quem envia os pacotes, no update do md5 em cada pacote que é enviado, sendo no final enviado um PDU do tipo CHECKSUM com o tamanho do hash criado, offset a 0, assim como o próprio hash. Do lado de quem recebe, é calculado o CHECKSUM do ficheiro recebido, é actualizado o md5 com o pacote recebido e é posteriormente verificado o hash obtido, sendo então possível saber se o ficheiro foi recebido com sucesso ou não, verificando assim a sua integridade.

Controlo de Fluxo: Como controlo de fluxo, decidimos implementar um sistema com *Data Blocks* de 100 pacotes de dados. Basicamente, o objetivo é que em caso de que o tamanho do ficheiro a enviar seja grande o suficiente para que se tenham de enviar mais que 100 pacotes de dados, apenas seriam enviados inicialmente 100 pacotes, sendo logo após este envio, verificado se todos os pacotes foram recebidos, com recurso ao recebimento de ACK's por parte de quem está a receber os pacotes de dados. Para verificar se algum pacote não chegou ao destino, o sistema vai buscar todos os pacotes que ainda permaneçam dentro da coleção de pacotes enviados, uma vez que se estão aqui dentro, significa que ainda não foram acknowledged por quem está a receber. Uma vez obtidos os pacotes a retransmitir, estes são novamente

enviados e assim que todo o *Data Block* dos 100 pacotes sejam enviados com sucesso, o sistema passa ao envio dos próximos 100 pacotes e assim sucessivamente. No caso do número dos pacotes ser menor que 100, o sistema funciona do mesmo modo, com a diferença que não existirá próximo *Data Block* e o tamanho do *Data Block* será sempre 100 ou menor.

3.2 Testes e Resultados:

Para a construção dos testes, foram utilizados dois computadores na mesma rede sobre os quais incidiu a transferência dos ficheiros. Estes testes incidirão apenas nos testes sem erros provocados, uma vez que as imagens para comprovarem os testes com erros provocados faria com que o relatório excedesse o limite de páginas. De referir que na camada TransfereCC existe um if documentado com recurso ao uso da operação *Math.Random* no envio inicial de cada pacote, que faz com que sejam enviados 9 em cada 10 pacotes. Após serem enviados todos os pacotes do ficheiro a primeira vez, ou após ser atingido o limite máximo de pacotes por bloco a enviar, estes pacotes perdidos seriam então reenviados, sendo a sua ordem mantida com o uso do offset. Passemos então à demonstração da transferência dos ficheiros

Testes sem Erros Provocados Nesta secção, foram produzidos os testes para a transferência de ficheiros sem qualquer tipo de erro. Inicialmente é introduzido o seguinte comando no computador com o endereço 192.168.1.75:

java -jar tp2cc.jar wait 4000

Neste momento, temos um dos lados à escuta na porta 4000, esperando receber um comando do tipo "get"ou do tipo "put", para receber ou para enviar o ficheiro que será selecionado do outro lado.

Do outro lado foram introduzidos dois comandos em separados:

java -jar tp2cc.jar put ficheiroteste.txt 192.168.1.75 9000 4000
java -jar tp2cc.jar get ficheiroteste.txt 192.168.1.75 9000 4000

Este comando recebe tal como referido, o tipo de comando, se "put"se "get, o nome do ficheiro, o endereço para onde pretendemos transferir o ficheiro, a porta local e por fim a porta remota. De seguida mostramos os prints com o resultado das transferências. Optamos por transferir ficheiros pequenos para que consigamos mais facilmente avaliar todo o processo, no entanto, se fosse utilizado um ficheiro de tamanho muito maior, o resultado seria na mesma uma transferência bem sucedida.

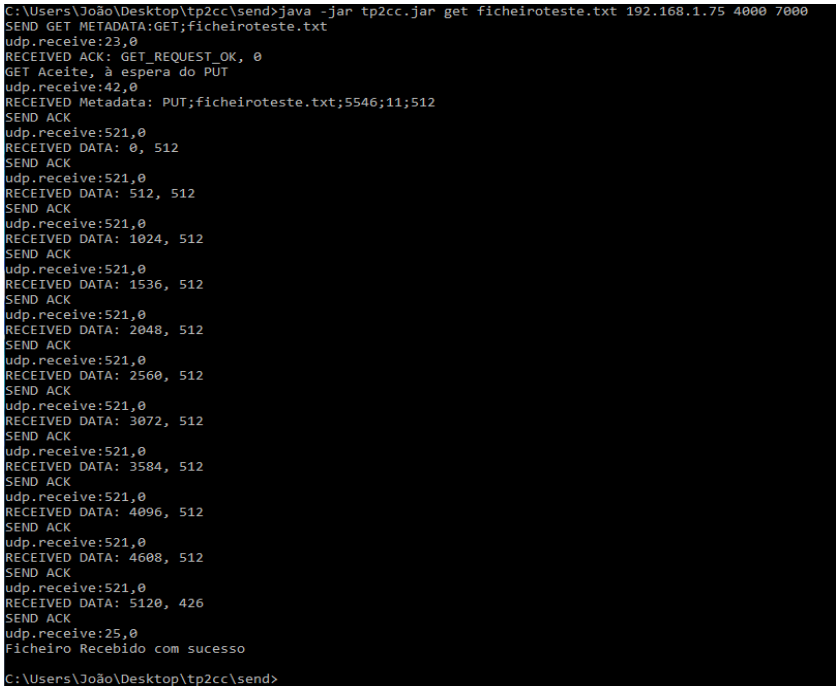

Figura 3: GET de um ficheiro.conf

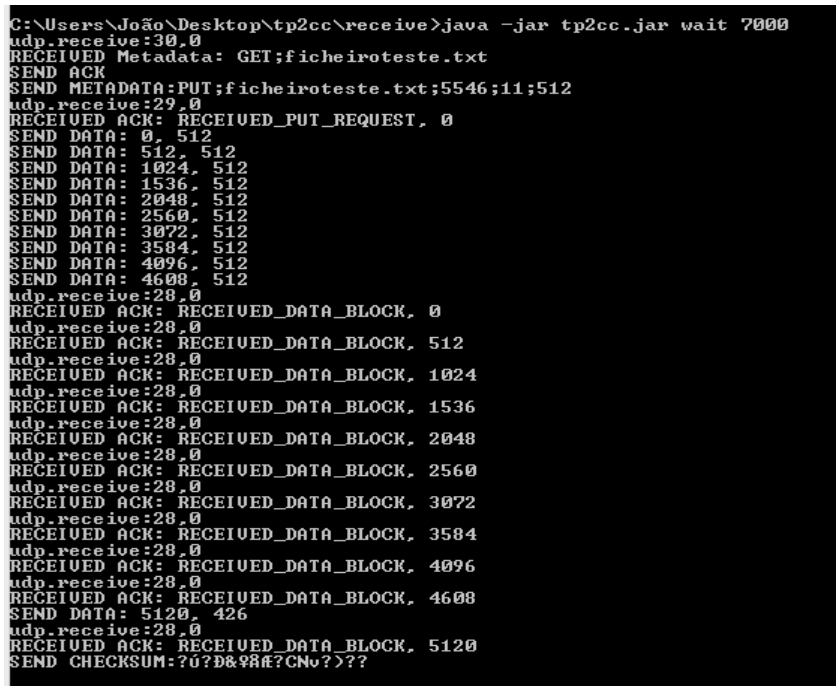

Figura 4: Pedido de GET de um ficheiro.conf

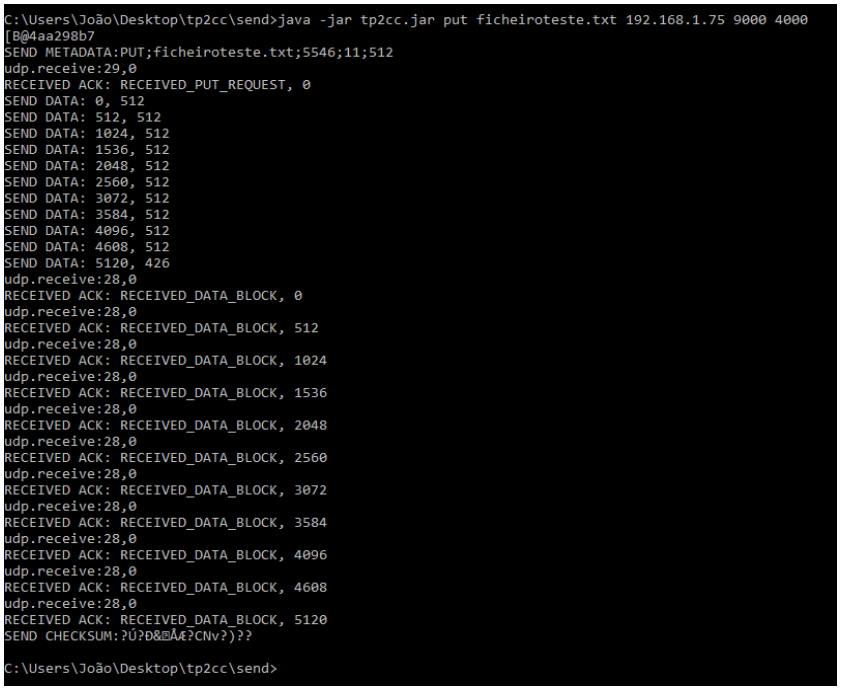

Figura 5: PUT um ficheiro.conf

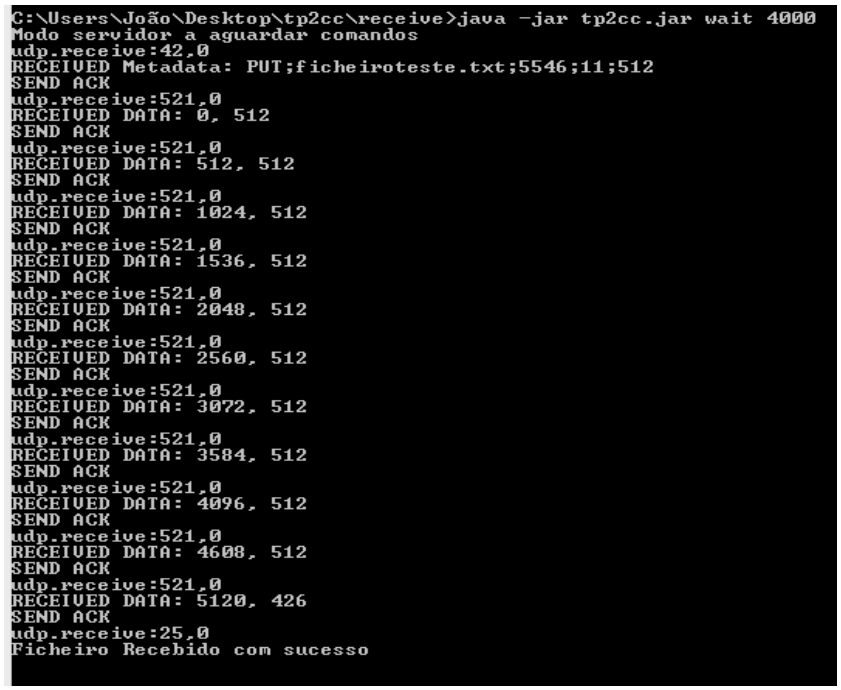

Figura 6: Pedido de PUT de um ficheiro.conf

4 Conclusão

Após a conclusão deste projeto podemos afirmar que temos um maior conhecimento sobre as transferências que são realizadas no dia-a-dia. Não fazíamos ideia que existiam tantos processos e informações por trás de uma transferência de dados, algo que é feito milhões de vezes diariamente por todo o mundo, e que tem de ser feito de uma forma rápida, viável e segura. É necessário haver um equilíbrio entre estes três requisitos de forma a oferecer a todos os utilizadores uma experiência o mais fluída possível sem comprometer, tanto na segurança como na integridade dos ficheiros.