Serviço de transferência rápida e fiável de dados sobre UDP

Luís Alves e Rafaela Rodrigues

Universidade do Minho, Departmento de Informática, 4710-057 Braga, Portugal
{a80165,a80516}@alunos.uminho.pt

Resumo Este relatório começa por apresentar a especificação formal do protocolo desenvolvido para um serviço de transferência rápida e fiável de dados sobre UDP. De seguida são feitas considerações sobre a implementação e termina-se com resultados obtidos utilizado o serviço.

1 Introdução

Este trabalho foi realizado no âmbito da Unidade Curricular de Comunicações por Computador, que se insere no 2º semestre do 1º ciclo de estudos do Mestrado Integrado em Engenharia Informática da Universidade do Minho.

Foi proposta a elaboração de um serviço de transferência fiável de ficheiros sobre uma conexão UDP genérica, sendo que esse serviço foi implementado recorrendo à linguagem de programação Java, sendo a versão base a 8.

Assim, o protocolo desenvolvido cumpre os requisitos base obrigatórios, que são:

1. Controlo de conexão
2. Identificação do ficheiro a transferir
3. Receção ordenada dos fragmentos do ficheiro
4. Controlo de erros, com retransmissão e verificação de integridade.

Cada um destes itens será abordado nas secções seguintes, bem como as funcionalidades extras, que são:

1. Encriptação dos dados enviados
2. Verificação da integridade final do ficheiro
3. Mecanismo de congestão

2 Especificação do protocolo

Para efetuarmos a transferência entre dois sistemas, definimos 3 tipos de mensagens distintas passíveis de serem trocadas entre eles: mensagens de controlo, de dados e de gestão. São as que se apresentam na subsecção seguinte.

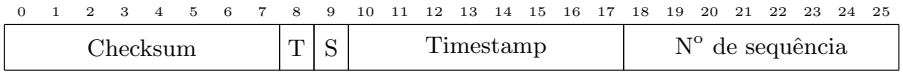

Figura 1. Formato base de um PDU

2.1 Formato das mensagens protocolares (PDU)

Existe um formato base a todas as mensagens trocadas no sistema, a que chamamos PDU. Contém os campos na forma que se apresenta na figura 1.

O campo T representa o tipo do PDU e o campo S o subtipo. O *Checksum* representa o valor computado do PDU total excluindo o próprio *Checksum*, a *Timestamp* o valor em milissegundos desde 1970-01-01T00:00:00Z, e o Número de sequência representa um identificador único do pacote.

Importante notar que o *Checksum* é calculado recorrendo à biblioteca do Java **CRC32**.

Para cada um dos 3 tipos de PDUs, definiram-se os valores presentes na tabela 1 para o campo de Tipo.

Tabela 1. Tipos de PDUs e respetivos valores

Tipo de PDU	**Valor**
Controlo	0
Gestão	1
Dados	2

Controlo Existem 4 tipos de mensagens de Controlo. São estes:

1. *Ack*
2. Pedido de conexão
3. Fim de conexão
4. Troca de chaves

Uma mensagem de *Ack* contém dois campos extra em relação ao PDU base. São estes o Número de sequência do pacote que pretende confirmar e uma mensagem (que poderá ser vazia ou de tamanho variável).

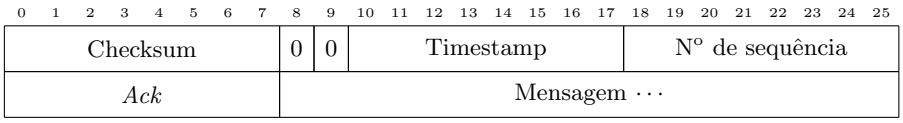

Figura 2. Formato de um *Ack*

Uma mensagem de Pedido de conexão apenas envia o Número de sequência a partir do qual o destinatário poderá validar os pacotes seguintes.

Uma mensagem de Fim de conexão serve para alertar o destinatário para dar início ao término de todos os recursos associados ao canal de comunicação estabelecido.

Já uma mensagem de Troca de chaves contém uma sequência de bytes de tamanho variável que representa uma chave (simétrica ou assimétrica).

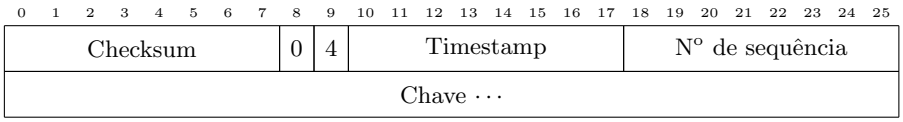

Figura 3. Formato de um *Troca de Chaves*

Para além disso, os valores associados a cada subtipo são os que se apresentam na tabela 2.

Tabela 2. Subtipos de Controlo e respetivos valores

Subtipo de PDU	**Valor**
Ack	0
Pedido de conexão	1
Fim de conexão	3
Troca de chaves	4

Gestão Existe apenas um subtipo de mensagem de Gestão: o identificador de ficheiro. No entanto, foi criado este subtipo de Gestão uma vez que o protocolo poderá ser alterado para suportar um outro tipo de PDU: listagem de ficheiros a pedido. No entanto, como não era um requisito fundamental para este trabalho, optamos por não incluir o seu desenho e implementação, apesar de ter sido definido um tipo capaz de albergar esse tipo de funcionalidade.

Assim, um PDU do tipo de Gestão inclui também um *byte* com a direcionalidade do pacote: se for 0, então é porque se trata de um *download*. Se for 1, então é porque se trata de um *upload*.

Uma mensagem de Identificação de Ficheiro possui um identificador do ficheiro, cujo tamanho é variável. Um exemplo deste campo é o seguinte "./programa-que-desejo-transferir.txt". Na figura 4, o campo D representa a direcionalidade do pedido. Se for de *upload*, indica que deseja fazer *upload* do ficheiro no campo identificador de ficheiro. Se for de *download*, que deseja fazer *download* desse ficheiro.

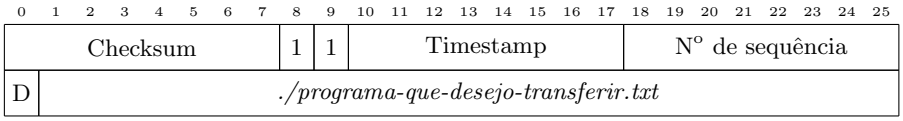

Figura 4. Formato de um Identificador de Ficheiro

Dados Existem dois subtipos de mensagens de Dados: Primeiro Bloco e Outro Bloco.

Uma mensagem do tipo Primeiro Bloco contem um datagrama da forma presente na figura 5, e cujos campos são explanados na tabela 3.

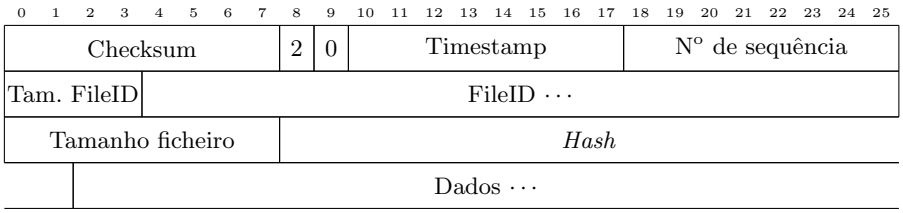

Figura 5. Formato de um Primeiro Bloco

Tabela 3. Campos do Primeiro Bloco

Campo	**Valor**
Tam. FileID	Tamanho do campo FileID
FileID	Identificador do ficheiro
Tamanho ficheiro	Tamanho total do ficheiro a transferir
Hash	Valor de *hash* do ficheiro completo
Dados	512 primeiros *bytes* do ficheiro

Relativamente a uma mensagem do tipo Outro Bloco, esta contem um datagrama tal como o da figura 6. É muito similar ao Primeiro Bloco, mas não contém os campos tamanho do ficheiro nem a *hash* deste, uma vez que essa informação já foi transmitida no Primeiro Bloco. Os campos associados a este datagrama encontram-se detalhados na tabela 4.

Uma outra diferença relevante entre o Primeiro Bloco e o Outro Bloco, é que o primeiro apresenta um tamanho máximo fixo de dados a transmitir, que é de 512 *bytes*, enquanto que o segundo pode apresentar um tamanho máximo de dados variável até 32000 *bytes*.

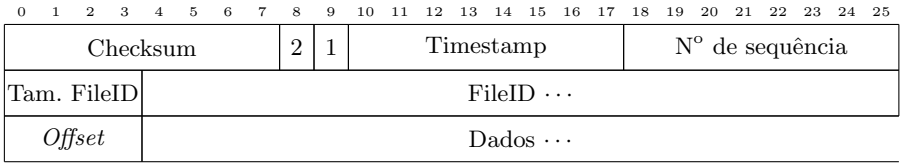

Figura 6. Formato de um Outro Bloco

Tabela 4. Campos do Outro Bloco

Campo	**Valor**
Tam. FileID	Tamanho do campo FileID
FileID	Identificador do ficheiro
Offset	Posição a partir da qual estes dados pertencem no fichiro
Dados	Quantidade variável de *bytes* do ficheiro

2.2 Interações

De forma a ser possível identificar os tipos de datagramas que foi necessário desenvolver, elaboramos um diagrama de sequência representativo de uma típica transferência de ficheiros, usando todas as funcionalidades previstas. É o que se apresenta na figura 7.

Assim, são evidentes 3 fases principais no serviço: o estabelecimento da conexão, a transferência do ficheiro, e o término da conexão.

Na primeira fase, o utilizador envia um pedido ao servidor para este iniciar uma nova sessão. Este terá de responder afirmativamente (se tudo correr bem) e essa resposta é do tipo Troca de chaves, uma vez que inclui a chave pública RSA do servidor. De seguida, o cliente utiliza essa chave pública para encriptar uma chave AES gerada por ele próprio. Assim, responde ao servidor com um novo datagrama de Troca de chaves, mas que contém a chave simétrica encriptada com a chave pública do servidor. Quando este recebe o datagrama, desencripta a chave simétrica recorrendo à sua chave privada e passa a comunicar com o servidor encriptando todos os seus datagramas com a chave simétrica recebida pelo utilizador.

Um problema evidente neste estabelecimento seguro da conexão é a não validação da chave pública do servidor. Isso poderá comprometer o serviço a ataques que simulem a identidade do servidor.

A fase seguinte é a de transferência de ficheiro. Findo o estabelecimento de conexão, o servidor fica à espera de um datagrama de Gestão, que indique qual a ação a executar. Para efeitos demonstrativos, assumamos que é um datagrama de pedido de *download* (o processo é similar quer para *upload*, quer para *download*, apenas é alterada a direcionalidade dos datagramas). Ao receber o datagrama de Identificação do Ficheiro, o servidor responde com o Primeiro Bloco, um datagrama de Dados. Este datagrama contém informações importantes acerca do ficheiro a ser transferido, nomeadamente o seu tamanho final e o valor de *hash* computado com recurso ao algoritmo *SHA-1*. Após receber a confirmação

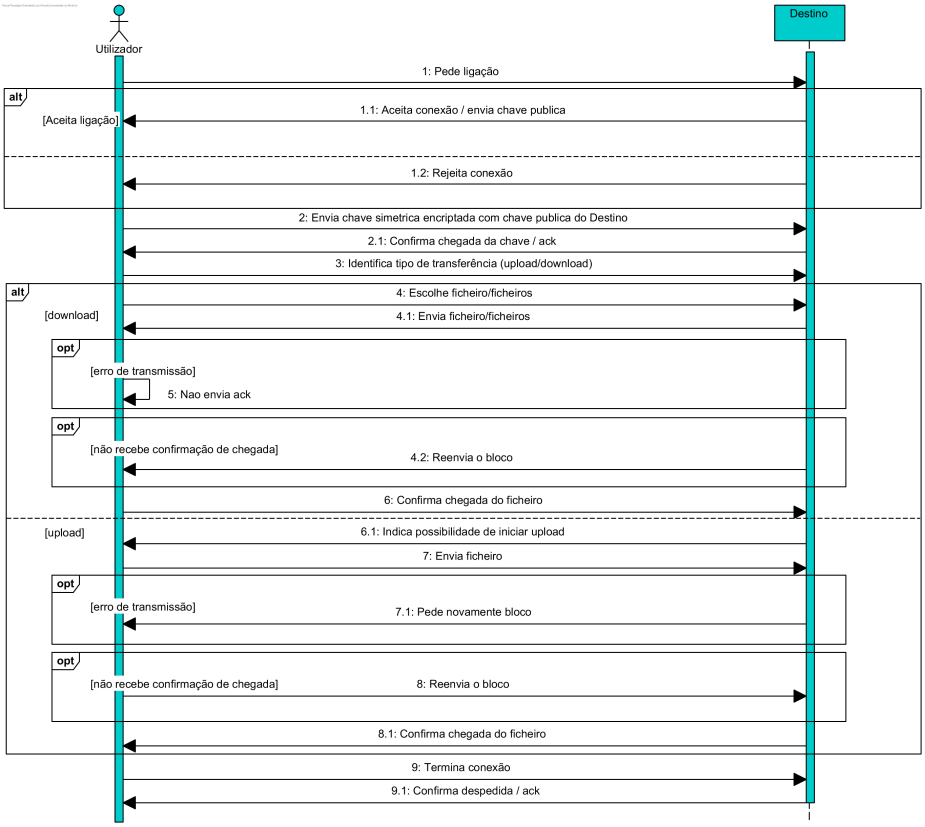

Figura 7. Diagrama de Sequência

de chegada deste datagrama (isto é, após o cliente enviar um *Ack* relativo ao datagrama de Primeiro Bloco), o servidor poderá começar a enviar os restantes blocos, que serão confirmados atomicamente pelo cliente. Após o envio da última confirmação por parte do cliente, este enviará um datagrama de término de conexão, sendo que o servidor terá de responder com um *Ack* a este datagrama para ambos terminarem os recursos associados à transferência normalmente.

3 Implementação

Este protocolo e respetivas aplicações (cliente e servidor) foram implementados recorrendo ao Java 8, uma vez que era a linguagem de programação com a qual os membros do grupo estavam mais familiarizados. Assim, optou-se pela criação de 4 *packages*, que representam 4 módulos distintos da aplicação. São estes:

1. Agente UDP
2. *App*
3. *Security*
4. TransfereCC

O primeiro contém os diversos datagramas previstos para o protocolo, bem como classes que permitem o envio e receção de datagramas com esse formato.

O segundo é uma simples aplicação de terminal, que pode ser usada para iniciar um servidor ou um cliente, sendo que este último poderá ser utilizado com o seguinte formato: `java Client <PUT/GET> <filename> <serverIP>`.

O terceiro é um módulo dedicado à implementação do par chave pública-privada e chave simétrica.

O último, é dedicado aos mecanismos de transferência fiável recorrendo ao agente UDP como base para a transferência.

Relativamente a mecanismos implementados, cada datagrama enviado pelo protocolo é reenviado até 5 vezes caso não receba a resposta pretendida, e sempre que um bloco de ficheiro é enviado com sucesso à primeira tentativa, a próxima série de 4 blocos a serem enviados terá o dobro do tamanho no campo de dados. Por outro lado, caso haja alguma tentativa falhada, o tamanho da próxima série de 4 blocos enviados simultaneamente será reduzido para metade.

3.1 Estruturas de dados

Os *packages* mais desenvolvidos e com mais funcionalidades são o Agente UDP e o TransfereCC.

O primeiro possui uma classe *Receiver* e uma classe *Sender*, sendo que a primeira apresenta várias estruturas de dados que suportam o seu funcionamento.

Um *Receiver* possui uma lista de *DatagramPackets*, que serão posteriormente processados por uma classe privada do *Receiver*: o *Processor*. Como, para cada *Receiver*, será criada uma nova *Thread* com um *Processor*, o primeiro poderá continuar a receber pacotes UDP e a colocá-los na lista de *DatagramPackets*

que serão processados por uma *Thread* distinta, aumentando o tempo disponível para receber novos datagramas. Para além disso, possui um *HashMap* com chaves *SequenceNumber* e valor um *Ack*, para que *Threads* adormecidas à espera de um determinado *Ack* possam ser alertadas quando este chegar, através de *Conditions*.

Para além disso, tanto o *Receiver* como o *Sender* possuem um par de chaves do pacote *Security*, que lhes permitirá desencriptar ou encriptar os datagramas recebidos, de acordo com o estado da ligação.

Quanto ao TransfereCC, este apresenta dois pares de classes relevantes: o *Client* e *ClientState*, e o *Server* e *Session*.

O primeiro mantém uma série de variáveis que permitem calcular o *retransmission timeout*, usando como base o algoritmo especificado em RFC 2988. Para além disso, cada *ClientState* mantém informação sobre o ficheiro a ser transferido e guarda num *HashMap* os pedaços transferidos do ficheiro (sendo a chave o *offset*, e o valor um *array* de bytes do ficheiro). Para além disso, também guarda a *Hash* do ficheiro e o seu tamanho final.

Já o *Server*, mantém uma *Session* para cada cliente que se conecta. Cada *Session* tem estruturas similares ao *ClientState*, uma vez que mantém o estado da ligação, possui um conjunto de chaves que são usadas para a comunicação (tal como o *ClientState*, e mantém num *HashMap* os pedaços de ficheiro transferidos para o cliente.

3.2 Métodos

No *package* Agente UDP, os métodos principais são o `sendDatagram` da classe *Sender*, que permite o envio de um datagrama para um determinado endereço IP, e os métodos de `get` da classe *Receiver*, que aguardam um determinado período em milésimos de segundo até devolverem o tipo de PDU desejado, ou *null*.

No *package* TransfereCC, o *Client* permite estabelecer uma conexão com o servidor através do método `startConnection`, permite requisitar um ficheiro através do método `requestFile(String nomeFich)` e permite terminar uma conexão com o servidor através do método `endConnection`. Já o Servidor, poderá apenas receber pedidos de conexão através do método público `receive ConnectionRequest`.

Para além disso, existe também uma classe, de nome *Reliable Piece* que serve para enviar um pedaço de ficheiro numa *Thread* única, retransmitindo até 5 vezes se não receber o *Ack* do datagrama enviado no *timeout* definido.

4 Testes e resultados

Relativamente a testes efetuados, o grupo não conseguiu testar a aplicação no *core*, uma vez que a versão do Java utilizada era superior àquela que lá se encontrava. No entanto, a transferência foi testada entre computadores distintos na mesma rede local, bem como no próprio computador. Para além disso, também

foi colocada a *Thread* principal de execução do programa a dormir durante fases da transmissão de forma a simular perda de pacotes, e o sistema conseguiu recuperar como expectável: caso o outro lado precisasse de reenviar mais do que 5 vezes, a transferência terminava. Caso contrário, retomaria a transmissão.

Por exemplo, a transmissão de um ficheiro com 4013 bytes demorou cerca de 400ms, mesmo imprimindo para o ecrã algumas informações relevantes sobre o estado da transferência.

5 Conclusões

Tendo em conta o trabalho realizado, o grupo considera que os objetivos mínimos foram atingidos. No entanto, a modularização feita permitia que fossem desenvolvidas mais funcionalidades que o enriqueceriam, mas consideramos que o benefício na classificação final não refletiria o trabalho extra a ser desenvolvido. E esse é um dos pontos menos positivos deste trabalho. Tendo em conta a cotação atribuída a este, pensamos que o trabalho pedido e sugerido para uma classificação boa era excessivo. Ainda assim, o trabalho em si foi desafiante e permitiu consolidar conhecimentos sobre a camada de transporte, quer de TCP quer de UDP.