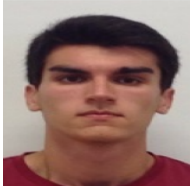
Comunicações por Computadores: TP2
Serviço de transferência rápida e fiável de dados sobre UDP

Universidade do Minho

Mestrado Integrado em Engenharia Informática
Grupo 410

José Ramos
A73855



Rafael Silva
A74264

Conteúdo

Lista de Figuras

1 Introdução

No presente relatório vamos abordar a resolução e verificação do projeto do segundo trabalho pratico realizado no âmbito de Comunicação por Computadores. Podemos então encontrar uma analise das medidas que utilizamos para resolver o problema proposto no enunciado , bem como todas as classes , funcionalidades e recursos que foram necessários implementar ao longo do desenvolvimento do projeto.

2 Especificação do protocolo

2.1 Formato das mensagens protocolares (PDU)

Antes de pegar no computador e começarmos a escrever código para criarmos um sistema de transmissão de ficheiros baseado em UDP, fiável e seguro, decidimos discutir qual seria o formato do PDU que iria ser transmitido de uma ponta do serviço para a outra de modo a abordarmos os tópicos pretendido e definirmos já uma base do projeto que iríamos desenvolver. Este *Protocol Data Unit* define-se em dois segmentos:

Figura 1: Desenho de PDU

- *Header* : onde guarda a informação necessária ao bom funcionamento e controlo da transferência quer pelo lado recetor quer pelo que envia.
- *Data Unit* : onde se encontra os dados a serem transmitidos no dado *PDU*

2.1.1 Header

O nosso header representa-se em 24 bytes divididos entre a vária informação que transmite, onde a ordem de leitura é de grande importância, nomeadamente :

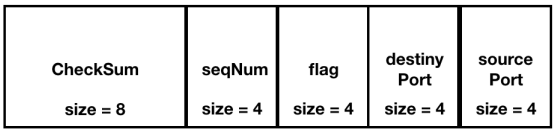


Figura 2: Desenho de Header

- Checksum(8 bytes) : Valor utilizado para verificarmos a integridade do PDU relativo ao controlo da informação da Data Unit
- seqNum(4 bytes): Numero de sequencia do pacote, que permite o controlo da transferência e a reorganização e reestruturação dos dados no fim da transferência.
- flag(4 bytes):
- destinyPort (4 bytes) : Endereço de Destino(4 bytes) que deve receber o PDU , assim uma dada maquina consegue verificar se é o destinatário do pacote de dados.
- sourcePort (4 bytes) :Endereço de Envio(4 bytes) :porta que representa quem enviou o PDU , permite á maquina recetora verificar se recebeu a informação da fonte correta, a qual verifica com o valor que recebeu quando iniciou a ligação para transferência.

2.1.2 Data Unit

Informação a ser transmitida pelo pacote de dados, será montada com todos os as Data Units recebidas em função do seu numero de sequencia, tal que no fim seja o Ficheiro que pretendemos transmitir.

2.2 Métodos de Controlo da transferência implementados

2.2.1 Funcionalidades Básicas

- *Identificar o ficheiro ou ficheiros a transferir, download ou upload, com nome local e opcionalmente um nome remoto;*

  No inicio da transferência definimos na maquina que envia o ficheiro a enviar e o nome que este terá ao chegar a maquina recetora.

- *Garantir uma receção ordenada (caso os segmentos cheguem fora de ordem, deve ser possível ordena-los de forma correta);*

  O envio das mensagens PDU ocorre de forma ordenada, com a ajuda de seqNum no Header que permite controlar a ordem de envio e receção, bem como futura reposição do ficheiro completo no fim da transferência.

- *Controlo de erros, com verificação de integridade e retransmissão se necessário (retransmitir pacotes perdidos ou em erro);*

  O projeto possui vários métodos de controlo de erros, nomeadamente , Checksum no Header de cada PDU, bem como controlo do numero de sequencia que permite captar falta de algum PDU. Em caso de erro a maquina que esta a enviar , não recebe no seu receiver o ack em falta e envia a informação para o seu Sender do erro, levando a que haja retransmissão.

2.2.2 Funcionalidades Adicionais

- Controlo de fluxo
  Este aspeto do programa resolvemos recorrendo ao protocolo de Janela Deslizante ("Sliding Window Protocol") que consiste no envio e ACK tal que o emissor começa por enviar um número de pacotes w - "tamanho da janela". O tamanho da janela é o número de pacotes que podem ser enviados sem qualquer ACK do receptor. E a cada Ack que recebe a janela desliza uma unidade permitindo o envio de outro PDU sem a receção de ack do PDU anterior, pois temos o processo de Sliding Window a ocorrer.

- Controlo de congestão

  O programa aplica um protocolo de AIMD (Aditive Increase Multiplicative Decrease) que controla o congestionamento condicionando o tamanho da window dependendo da eficiência da transferência e da quantidade de PDU enviados com êxito.

3 Implementação

Para a execução deste trabalho iremos usar várias classes, que irão ser explicadas na secção em baixo.
Estas irão se dividir em duas secções:

- **AgenteUDP**
- **TrasnfereCC**

Irá seguir a seguinte estrutura/arquitetura:

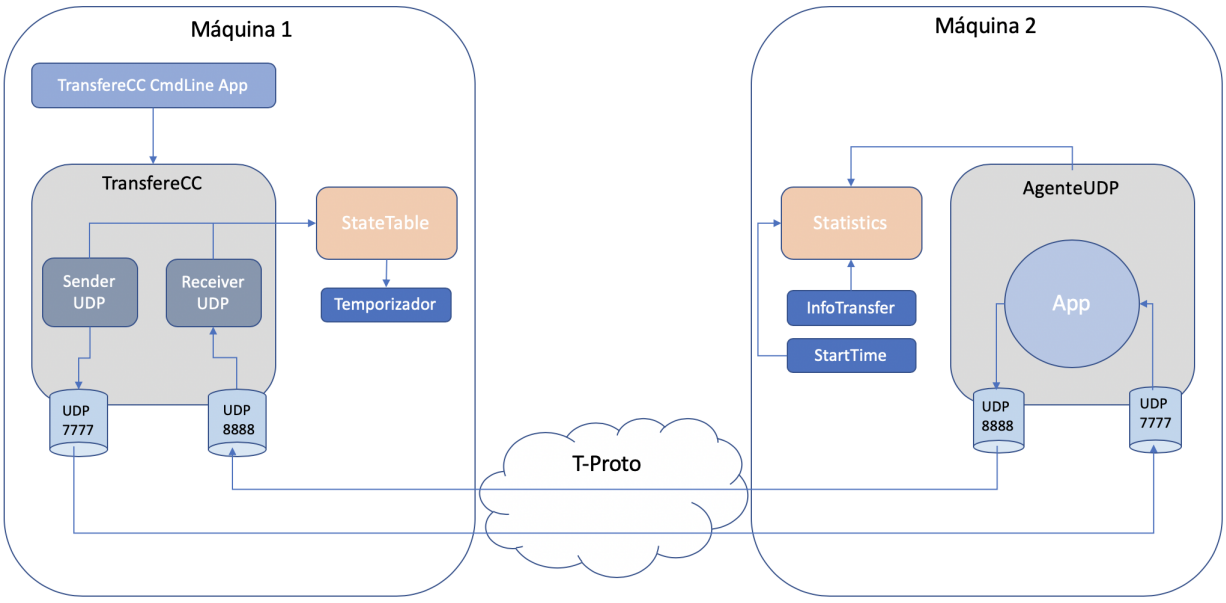


Figura 3: Arquitetura da aplicação desenvolvida

3.1 Estruturas dados

Para a execução deste trabalho iremos usar várias classes que foram mostradas na imagem em cima.

3.1.1 AgenteUDP

Esta classe tem como função receber os pacotes enviados para a porta **UDP-7777** e fazer o tratamento dos mesmos e de seguida irá enviar um **ACK** para a porta **UDP-8888**.
Para esse efeito esta recorrerá a métodos e classes auxiliares para esse efeito, que irão ser explicados de seguida.

- Iremos definir as variáveis necessárias para ser intuitiva a implementação:

```
private static final int HEADER = 24;
private static final int PACK_SIZE = 1000 + HEADER;
private static final int SERVER_PORT = 7777;
private static final int ACK_PORT = 8888;
private final String pathDestiny;
private final InfoTransfer info;
private StartTime time;
```

- Primeiro o mesmo AgenteUDP é corrido no terminal da máquina que irá receber o ficheiro, este solicita ao utilizador que introduza um path destino para onde pretende guardar o ficherio.
- De seguida o mesmo irá executar a função *startAgenteUDP()*, que tem como propósito fazer um controlo da transferência em questão, a mesma irá usar *DatagramSocket* e *DatagramPacktet*, o mesmo irá esperar até que aconteça alguma conexão com o mesmo. Quando existe uma conexão ele irá usar o *InetAdress* e a função *receive* da biblioteca de *DatagramSocket* para a receção do pacote.
- De seguida o mesmo irá confirmar o *CheckSum* do pacote recebido, e das suas respetivas *flags*, *porto de destino* e *porto de origem*. Isto serve para confirmarmos se o pacote recebido está correto e não está corrompido. Neste caso as *flags* irão tomar vários valores que serão explicados alguns casos de seguida.
- Em seguida o mesmo irá verificar a *flag* do pacote recebido e irá certas instruções consoante a mesma:
  - ***flag==0*** - Pacote não é o último.
  - ***flag==1*** - É o último pacote.
  - ***flag==2*** - Pacote duplicado.
  - ***flag==3*** - Pacote corrompido
  - ***flag==4*** - É o primeiro pacote.
  - ***flag==5*** - Origem ou destino desconhecidos.

- Iremos ter várias verificações da informação do pacote que levarão tomar a diferentes situações:
  - Caso o pacote esteja corrompido este irá ignorar esse pacote e irá verificar se o porto de destino ou o porto de origem estarão errados, caso estejam este irá cancelar a transferência, enviando um *ACK* com a *flag* **5**, caso não seja esse o problema ele irá enviar um *ACK* com a *flag* **3** e o número de sequência em questão, isto indica-nos que o pacote em questão está corrompido e pede o reenvio do mesmo.
  - Caso o pacote não esteja corrompido este irá verificar se o pacote em questão foi recebido por ordem, caso o mesmo nao tenha sido, este irá mandar um *ACK* com a *flag* **2** e o número de sequência em questão, para o pacote não seja reenviado. Se o pacote em questão foi recebido na correta ordem este irá ser verificado se é o último pacote ou não, caso seja irão ser enviados 20 pacotes **ACK** para finalizar a conexão, com o número de sequência **-2** e *flag* **1**. Se não for o último pacote o mesmo escreve a informação do mesmo no ficheiro em questão.
- A função responsável pela geração dos pacotes **ACK** é denotada:

```
private byte[] generatePack(int ackNum, int flag){
    ...
}
```

- Esta classe utilizada duas classes auxiliares para fins estatísticos da transferência, denotadas **InfoTransfer** e **StartTime**.

3.1.2 TransfereCC

Esta classe tem como função receber os pacotes enviados para a porta **UDP-8888** e fazer o tratamento dos mesmos e de seguida irá enviar um **ACK** para a porta **UDP-7777**.
Para esse efeito esta recorrerá a métodos e classes auxiliares para esse efeito, que irão ser explicados de seguida.
Irá utilizar para o envio de pacotes uma **Thread** denotada **SenderUDP** e irá utilizar outra **Thread** denotada **ReceiverUDP** que será responsável pela receção de pacotes.
Também irá utilizar uma classe **StateTable** que será utlilizada para o controlo da transferência.

- O primeiro passo é fornecer o IP para o qual pretendemos enviar o ficheiro (tem de ser o mesmo onde o AgenteUDP esta a correr) e o seu caminho correspondente. E de seguida serão criadas as duas threads para o tratamento da conexão.

- A **Thread** denotada **SenderUDP** irá enviar os pacotes com a informação para o **AgenteUDP**, para esse efeito está irá recorrer a classe **StateTable** para fazer o controlo do envio e em seguida irá proceder ao envio do pacote que foi criado usando a seguinte função denotada de:

```
private byte[] generatePack(int seqNum, byte[] dataBytes, int flag){
    ...
}
```

- A **Thread** denotada **ReceiverUDP** têm como função receber os pacotes enviados pelo **AgenteUDP**, para fazer o controlo da transferência, mudando certos valores da **StateTable** para esse efeito.

- Esse controlo é baseado no método de **Janela Deslizante**, noutro método denotado **Go Back-N** e por fim no controlo de congestão.

- A classe **StateTable** irá usar a seguinte estrutura:

```
private static final int HEADER = 24;
private static final int PACK_SIZE = 1000;
private static final int WINDOW_SIZE = 20;    // Window starting Size
private static final int TIMER_VALUE = 300;   // 300ms until timeout
private static final int SERVER_PORT = 7777;
private static final int ACK_PORT = 8888;
private static final int STARTING_WINDOW = 10;
private static final int MAX_WINDOW_SIZE = 20;  // Max size da Window
private static final int ADDITIVE_FACTOR = 1;   // fator aditivo de AIMD
private static final int DIVISIVE_FACTOR = 2;   // fator divisor de AIMD
private int baseWindow;     // Numero em que esta a base da janela
private int proxNumSeq;     // Próximo numero de sequencia na janela
private List<byte[]> listPacks; // Lista de pacotes enviados
private Timer timer;
private boolean downloadComplete;
private String pathSource;
private DatagramSocket socketSender, socketReceiver;
private String host;
private String command;
private int windowdegree ; // grau ou tamanho da janela
private boolean packetLoss; // indicador se ocorreu packetLoss
private int packetLossCount; // numero de pacores perdidos
```

Irá usar uma classe denotada **Temporizador** que nos indicará se na conexão aconteceu *TimeOut*.
Esta classe tem os seus métodos que serão utilizados pelas duas **Threads** em cima explicadas.

Com a utilização das classes em cima expostas o **TransfereCC** é capaz de realizar uma transferência com sucesso.
Usando os métodos correspondentes para cada operação concreta.

3.2 Bibliotecas de suporte usadas

Fazendo uma análise pelo código implementado é possível destacar o seguinte conjunto de bibliotecas de funções que foram utilizadas:

- **java.net.DatagramSocketejava.net.DatagramPacket** - Utilizamos as funções destas bibliotecas para criar corretamente os socketsUDP e os correspondentes packets para fazer a troca de informação através de sockets UDP.
- **java.net.InetAddress** - Esta biblioteca foi usada para obter o endereço para onde se está a efetuar a transferência.
- **java.io.File, import java.io.FileWriter** - Estas bibliotecas foram utilizadas com o intuito de criar um ficheiro para a escrita da informção, bem com a leitura da mesma.
- **java.util.TimerTask** - Foi utilizada a implementação do Temporizador de TimeOut nas Threads.
- **java.util.zip.CRC32** - Esta biblioteca foi utilizada para efetuar o *CheckSum* dos pacotes.
- **import java.nio.ByteBuffer** - Foi utlizada para a alocação do array do pacote.

4 Testes e resultados

Após serem efetuados vários testes, quer na máquina *Core* com a topologia fornecida para as aulas práticas quer na máquina nativa, podemos verificar que na nossa aplicação faz o que é pedido e transfere com sucesso um ficheiro de um lado para o outro, baseado em UDP.
Podendo visualizar a troca de ACK entre o AgenteUDP e o TransfereCC e no final podendo visualizar as estáticas da transferência que foi efetuada.

5 Conclusões

Ao longo do projeto sentimos aumentar o nosso conhecimento e experiência na área da Comunicação por Computadores, pois enquanto este se encontrava em fase de desenvolvimento fomo-nos deparando com diverso problemas, erros e mesmo novas oportunidades de implementar outras funcionalidades.

A titulo de exemplo o *Protocol Data Unit* que apresentamos no trabalho não é o mesmo que desenhamos e idealizamos na primeira aula, pois ao longo do projeto apresentaram-se algumas dificuldades que não tínhamos antevisto mas que permitiram enredar em novas funcionalidades tal que chegamos ao protocolo que apresentamos em cima.

Deste modo concluímos que embora o sistema de transferência que desenvolvemos não chegue a ser tão fiável nem completo como o TCP , é uma opção mais segura e eficaz que o UDP.