Comunicações por Computador — Trabalho Prático 2

Desenho e Implementação de um Jogo Distribuído na Internet

Rui Camposinhos, Carlos Rafael Antunes, and Nuno Oliveira

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal
e-mail: {a72625, a67711, a67649}@alunos.uminho.pt

Resumo O presente relatório descreve a implementação de um jogo multi-utilizador online, com questões sobre músicas reproduzidas em tempo real. Foi implementada uma arquitectura com um servidor e multi-utilizadores, conectados numa rede local e com troca de datagramas UDP.

1 Introdução

No presente trabalho pretende-se implementar um jogo multi-utilizador online, com questões sobre músicas reproduzidas em tempo real.

Com base nos requisitos definidos por [1], o sistema deverá ser distribuído, com comunicações TCP entre servidores e comunicações UDP entre clientes e servidores, que procure optimizar a utilização da largura de banda, por transmissão selectiva, fiável e com controlo de fluxo e de erros (figura 1).

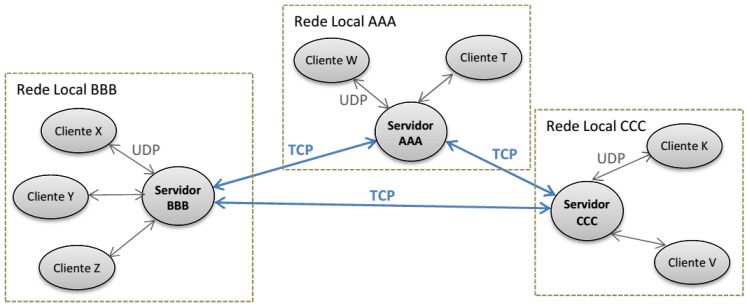

Figura 1. Arquitetura do sistema a implementar (de acordo com o enunciado [1])

2 Hipóteses Alternativas ao Enunciado

De uma forma geral foram adoptadas todas as hipóteses referidas no enunciado, salvo raras excepções que seguidamente se detalham:

- No ficheiro de base de dados foi adoptado um separador diferente para os campos. Alternativamente à vírgula (",") foi adoptado um ponto e vírgula (";"), de forma a evitar conflitos com o texto que pudesse incluir vírgulas.
- Nos tipos de pedidos dos clientes para os servidores, foi ignorado o pedido `RETRANSMIT`, tendo-se em alternativa implementado um pedido `TRANSMIT`, com o número da resposta pretendido.

– No pedido ACCEPT_CHALLENGE é devolvida a data e hora do desafio, em alternativa a um simples OK.
– Foram acrescentados dois campos aos tipos de resposta dos servidores para os clientes: Campo 19 - definido como uma etiqueta (tag) do PDU para dados gerais; Campo 30 - definido com NEXT_PACKAGE (pacote esperado).

3 Implementação

Todo o código desenvolvido para o presente projecto foi realizado com a linguagem de programação *java*.

Foram utilizadas várias bibliotecas auxiliares, sendo de destacar a biblioteca java.net e em particular a classe DatagramPacket [1], utilizada para criar conexões do tipo UDP.

3.1 Arquitectura do Sistema

Para o presente projecto foram desenvolvidas duas aplicações:

1. Uma aplicação para clientes, designada TP2-CC-MusicClient.
2. Uma aplicação para servidores, designada TP2-CC-MusicServer.

Não foi possível concluir a etapa de implementação das ligações TCP multi-servidor. Assim, a arquitectura do sistema implementada permite criar uma única instância de servidor (uma única rede local), com vários clientes a este conectados. As ligações cliente/servidor são todas realizadas por UDP.

3.2 Estruturas de Dados

As principais estruturas de dados implementadas foram: Campo, ListaCampos e PDU. Para tal foram criadas classes, com o mesmo nome, que seguidamente se detalham. A classe principal é a classe PDU, com definição de variáveis e tamanhos adequados a todos os campos que definem o datagrama.

Listing 1.1. Classe PDU

```
1 public class PDU {
2     private byte versao; /*codigo da versao - por omissao 0*/
3     private byte seguranca; /*seguranca - por omissao 0*/
4     private short label; /*identificacao do pedido*/
5     private byte tipo; /*codigo do tipo de pedido - ex:0->REPLY*/
6     private byte nCampos; /* numero de campos seguintes*/
7     private int tamanho; /*tamanho da lista de campos*/
8     private byte[] lista; /*lista de campos*/
9 }
```

A lista de campos, acima tratada como um array de bytes para uma maior abstracção, requer um maior detalhamento. Pelo exposto, foi criada a classe Campo e ListaCampos que permite codificar cada componente da lista de campos de forma correcta e sistemática. Posteriormente, na criação do PDU é feita uma conversão do ArrayList<Campo> para o array de bytes final pretendido.

Listing 1.2. Classe Campo

```
1 public class Campo {
2     private byte tag; /*numero do tipo de resposta*/
3     private int size; /* tamanho dos dados*/
4     private byte[] dados; /* array de dados*/
5 }
```

[1] http://download.java.net/jdk7/archive/b123/docs/api/java/net/DatagramPacket.html

Listing 1.3. Classe ListaCampos

```
1 public class ListaCampos {
2     private ArrayList<Campo> lista; /* lista com todos os campos da mensagem a enviar */
3 }
```

3.3 Outros Detalhes

De forma a controlar o fluxo entre servidor e clientes foi implementada uma estratégia de comunicação do tipo Stop-and-Wait, i.e., após envio de um pacote, o servidor espera confirmação do cliente para envio do pacote seguinte. Trata-se de uma técnica simples, mas que permite um controlo eficaz. De forma a ultrapassar as situações de erro (pacotes/confirmações perdidas) foi implementado um controlo por Time-Out no cliente. Após enviar uma confirmação, o cliente espera 20s pelo próximo pacote. Caso não o receba, reenvia confirmação de forma a forçar o reenvio do pacote perdido. Pode também optar por não reenviar e terminar o processo.

A escolha dos desafios pelo servidor é efectuada de forma aleatória. Sempre que é criado um desafio novo pelo cliente, o servidor escolhe de forma aleatória um dos desafios contidos na sua base de dados.

Para resolver o problema do limite de tamanho de um PDU foram criadas duas classes, uma no cliente e outra no servidor, que separam o PDU com tamanho excessivo em arrays de bytes de tamanho mais reduzido. Estes blocos são encapsulados em instâncias da classe PDU. A operação inversa de desencapsulamento e agregação também é realizada, como seguidamente se descreve. Os fragmentos do PDU original têm um campo que é o número do bloco, um campo com o byte 254 - informação de que não se trata do último bloco -, e um campo com o byte 19 - dados com o array de bytes do PDU original respectivo (PDU com tamanho excessivo). No cliente, os blocos do PDU são agrupados e os dados dos respectivos campos - com o byte 19 - são concatenados, regenerando o PDU original.

4 Testes e Resultados

A aplicação dos clientes é corrida em ambiente shell, com recurso a menus de opções. Seguidamente apresentam-se alguns outputs do cliente e servidor.

Inicializaçao do Cliente:

```
*** CC-Music ***

*** Menu Principal ***

1-Login
2-Registar
0-Sair

Opçao:
```

Registo de novo utilizador:

```
*** Registar ***

Introduza os seus dados:
Nickname: rui
Password: pass
Nome: rui pedro
Enviado!
Resposta recebida!
OK!
```

Resposta do servidor:

```
**Pacote de dados n.1 **

Versao correta.
Sem segurança.
Label: 1
Tipo: REGISTER
    Novo utilizador:
      Alcunha: rui
      Pass: pass
      Nome: rui pedro
Resposta enviada!

**Pacote de dados n.1 tratado **
```

Login do cliente:

```
*** Login ***


Nickname: rui
Password: pass
Enviado!
Resposta recebida!
Bem vindo rui pedro!
```

Resposta do servidor:

```
**Pacote de dados n.2 recebido **

Versao correta.
Sem segurança.
Label: 2
Tipo: LOGIN
    Login:
      Alcunha: rui
      Pass: pass
IP Recebido: /127.0.0.1 Port: 54029
Resposta enviada!

**Pacote de dados n.2 tratado **
```

Menu Principal Cliente:

```
*** CC-Music ***

*** Menu Principal ***

1-Criar novo desafio
2-Listar os desafios atuais
3-Entrar num desafio
0-Sair

Opçao:
```

Novo desafio:

```
*** Criar novo desafio ***

Introduza o nome do desafio: teste
Introduza a data (AAMMDD): 150608
Introduza a hora (HHMMSS): 180000
Enviado!
Resposta recebida!
Desafio criado!
À espera que o desafio começe...
Tempo de Espera: 443 segundos...
Tempo de Espera: 442 segundos...
```

Resposta do servidor:

```
**Pacote de dados n.4 recebido**

Versao correta.
Sem seguranca.
Label: 4
Tipo: MAKE_CHALLENGE
Desafio escolhido: 1
A ler o ficheiro desafio-000001.txt
Ficheiro lido com sucesso!
Novo desafio criado: teste
Resposta enviada!
**Pacote de dados n.4 tratado **
```

Envio da pergunta 1 do desafio pelo servidor:

```
**Pacote de dados n.5 recebido **

Versao correta.
Sem segurança.
Label: 5
Tipo: TRANSMIT
Pedido para a pergunta 1 do desafio "teste".
Pergunta: Quem canta esta cancao?
Respostas:
[Robert Smith, Bono Vox, Ninguem,
     e uma musica instrumental]
Resposta Certa: 3
Path da Imagem: 000001.jpg
Path da Musica: 000001.mp3
Tamanho da imagem: 105039
Imagem transformada em array de bytes!
Tamanho da imusica: 2411752
Musica transformada em array de bytes!
PDU criado!
A dividir em PDU's mais pequenos...
Número de pacotes: 52 Tamanho do PDU: 2516927
A guardar os pacotes na matriz...
Concluido!
Pacotes divididos com sucesso!
Resposta enviada!

**Pacote de dados n.5 tratado **
```

5 Conclusão e Trabalho Futuro

No presente trabalho foram discutidas as principais questões relacionadas com a implementação de uma comunicação servidor-cliente, por intermédio de UDP. Foram descritas as estratégias adoptadas, o controlo de erros e de fluxo, os tipos de estruturas de dados utilizadas e apresentados alguns exemplos de aplicação.

Devido a dificuldades relacionadas com o prazo de entrega do trabalho, não foi possível completar alguns dos objectivos propostos, designadamente:

- Não foi possível implementar conexões multi-servidores através de TCP.
- Não foi possível ultrapassar algumas dificuldades relacionadas com a partição dos PDUs de maior tamanho, por exemplo no envio de músicas.
- Não foram implementadas todos os tipos de resposta do servidor.

De referir também, que o protocolo de controlo de fluxo poderia ser melhorado através da utilização de uma janela deslizante, optimizando o número de mensagens trocadas entre servidor e cliente.

Acknowledgments

O presente trabalho foi realizado no âmbito da unidade curricular de Comunicações por Computador, ano lectivo de 2014/2015, da Licenciatura em Engenharia Informática, da Universidade do Minho.

Referências

1. Santos, A., Dias, B., Lima, M., Sousa, P.: Trabalho pratico no. 2 - desenho e implementacao de um jogo distribuido na internet (2015)