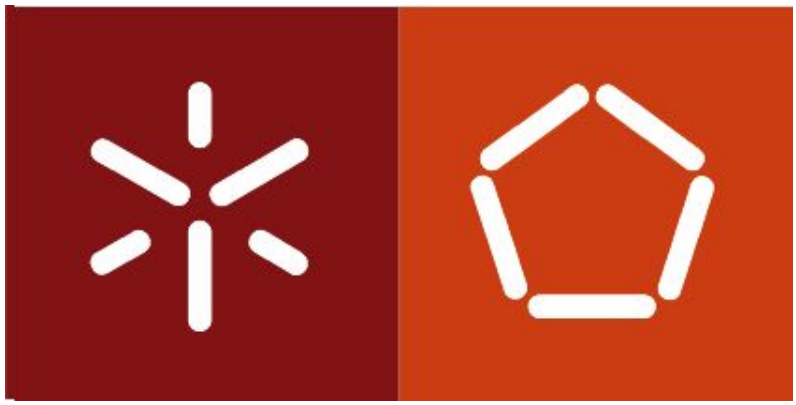

Universidade do Minho

Departamento de Informática

Mestrado Integrado em Engenharia Informática

Unidade Curricular de Comunicações Por Computador

2015/2016

Desenho e implementação de um sistema de partilha de áudio

André Geraldes a67673
Célia Figueiredo a67637
Gil Gonçalves a67738

Resumo

Neste relatório encontram-se descritas todas as etapas feitas e as decisões tomadas na criação de um sistema de partilha de áudio. Este sistema é composto por clientes, servidores e um servidor central. Os clientes fazem pedidos ao seu servidor local procurando uma determinada música, o servidor reencaminha o pedido a todos os clientes ligados, caso algum deles tenha a música ele informa o cliente que fez o pedido com os dados, ele liga-se por UDP ao cliente escolhido e faz-se a transferência da música. Caso nenhum cliente local tenha a música, é feito um pedido ao servidor central que reencaminha para os servidores ligados e faz os mesmos passos.

Introdução

Neste trabalho utilizamos os protocolos de transporte TCP e UDP.
As diferenças entre os dois residem no facto de o protocolo TCP ser um protocolo de de controlo de transmissão, ou seja, quando é enviado um pacote via TCP, existe a garantia que esse pacote é entregue. O protocolo TCP é um protocolo lento, visto que verifica se existe erros no pacote, se o pacote contiver erros tenta retransmitir o protocolo e é um protocolo bem comportado, ou seja não sobrecarrega a rede. Já o protocolo UDP é um protocolo que não se preocupa com os erros no pacote, ou seja, se um pacote tiver com erros ele entrega na mesma, pode acontecer entregar os pacotes sem a ordem que foi inicialmente transmitida e se a rede estiver congestionada ele envia o pacote na mesma.

Foi nos proposto trabalhar com estes dois protocolos nesta aplicação para podermos testar e ver todas estas características de cada um, e ainda com o desafio extra de otimizar o protocolo UDP.

Para enviar as músicas usamos o UDP visto que o UDP é um serviço sem conexão, pois não há necessidade de manter um relacionamento longo entre cliente e o servidor. Assim, um cliente UDP pode criar um socket, enviar um datagrama para um servidor e imediatamente enviar outro datagrama com o mesmo socket para um servidor diferente. Da mesma forma, um servidor pode ler datagramas vindos de diversos clientes, usando um único socket.

Diferenças e adições ao protocolo apresentado no enunciado

- Foi preciso adicionar mais tipos de PDUs para identificar certos pedidos, tipo 8 e tipo 9.
  - Tipo 8 foi utilizado como resposta a um pedido *REGISTER*;
  - Tipo 9 foi utilizado para fazer pedidos *CONSULT_REQUEST* entre servidores e servidor central.

Implementação

A implementação do sistema foi feita na linguagem Java e o desenvolvimento foi feito no IDE *NetBeans*. A forma de funcionamento da aplicação pode ser vista na figura 1.

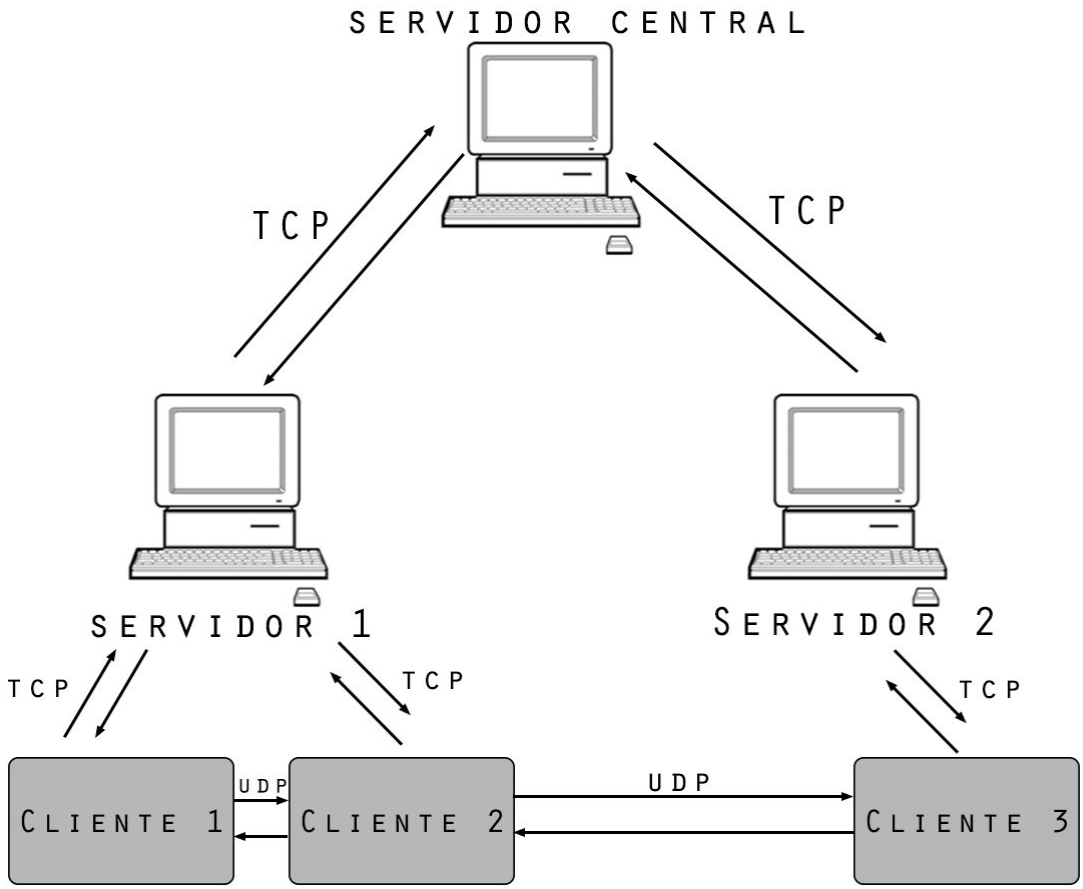

Figura 1. Funcionamento da aplicação.

E a arquitetura da aplicação é apresentada na figura 2.

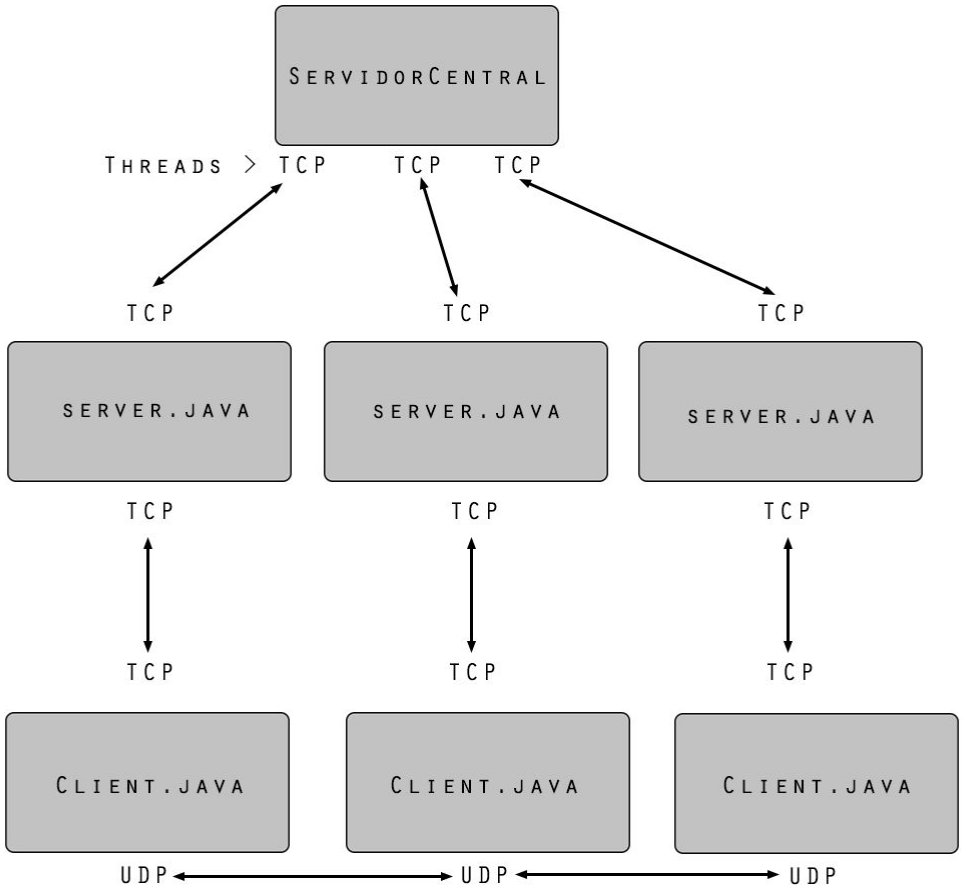

Figura 2. Arquitetura da aplicação.

Temos o limite de 1 servidor central. Limite de 1 servidor normal por máquina, os clientes podem ligar-se a um servidor da máquina ou de outra máquina, cada servidor pode ter vários clientes.

A conexão entre servidor e servidor central é feita por TCP assim como a conexão entre cliente e servidor. A conexão entre clientes é feita via UDP.

A aplicação é composta por 3 classes principais, *ServidorCentral.java*, *Server.java* e *Client.java*. Que são complementadas com as classes *PDU.java*, *User.java* e classes *Handler* para criar threads para tratar pedidos.

Classes

Vamos agora explicar cada classe.

- *ServidorCentral.java* - Este é o servidor central da aplicação, ao qual os outros servidores se conectam e registam obrigatoriamente. Este cria um

thread (*ServerHandler.java*) por cada servidor que se conecta, tratando cada pedido que recebe dele, registo ou procura de músicas.

- *Server.java* - Esta classe cria um servidor local ao qual se conectam e registam os clientes locais, é ele que guarda o registo de clientes ligados, trata de encaminhar pedidos entre clientes e pedidos ao servidor central. Tem uma thread (*ClientHandler.java*) por cada cliente.

- *Client.java* - Esta classe cria a interface ao utilizador (textual), primeiro faz o pedido ao utilizador para se identificar, envia o para o servidor e regista-se. Depois dá ao utilizador a opção de fazer uma consulta ou sair da aplicação. Tem duas threads, *ClientConsult.java* e *ClientUDP.java*, nas quais recebe pedidos de consulta do servidor e tratar da conexão UDP respetivamente.

- *PDU.java* - Classe que é usada para criar os PDUs respetivos, existe um método para criar cada PDU necessário e retornar o mesmo em bytes.

- *User.java* - Classe utilizada para guardar as informações de cada cliente.

Estas são as classes principais da nossa aplicação.

Funcionamento

- Funcionamento geral do TCP, exemplo.

```
private Socket centralSocket;

OutputStream osCentral = this.centralSocket.getOutputStream();
InputStream isCentral = this.centralSocket.getInputStream();

osCentral.write(value.getBytes());

byte[] response = new byte[256];
isCentral.read(response);
```

- Funcionamento do UDP, exemplo.

```
DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
```

```
this.serverSocket.receive(receivePacket);
...
DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
IPAddress, port);
serverSocket.send(sendPacket);
```

- Servidor Central

Ao iniciar abre um *socket* TCP na porta 4000, que fica à espera de conexões. Guarda num HashMap os servidores que estão registados.

```
public final static int DEFAULT_PORT = 4000;
private ServerSocket serverSocket;
...
this.serverSocket = new ServerSocket(DEFAULT_PORT);
...
while(true) {
        Socket s = this.serverSocket.accept();
        System.out.println("[+] Connection from " + s.getInetAddress());
        ServerHandler c = new ServerHandler(s, servers);
        new Thread(c).start();
}
```

Quando recebe uma conexão esta é tratada numa thread ServerHandler.java. Esta thread trata do registo de servidores e dos pedidos de consulta. O registo passa por receber um PDU de registo e guardar as informações dos servidores, caso corra bem informa o servidor que está registado, caso corra mal responde com um PDU que informa do erro.
Os pedidos de consulta recebidos são tratados por etapas:

- Verifica os servidores disponíveis, copia para um novo HashMap, remove o servidor que fez o pedido desta copia.
- Percorre num ciclo for todos os servidores a quem fazer pedido.
- Altera o tipo de pedido de 2 para 9.
- Conecta-se com o servidor e envia o pedido de *CONSULT_REQUEST*.
- Espera pela resposta e verifica qual foi a resposta.
- Caso a musica tenha sido encontrada, para o ciclo for, caso contrário continua.
- Após isto verifica se encontrou a musica e envia o PDU do tipo *CONSULT_RESPONSE* para o servidor que fez o pedido as informações, *FOUND(1)* ou *NOT_FOUND(0)*.

- Servidor

A sua porta TCP default é a 3000. Guarda num HashMap os clientes que estão ligados.

```
private HashMap<String, User> users;
public final static int DEFAULT_PORT = 3000;
```

Quando inicia cria um *serverSocket* onde irá receber os pedidos dos clientes. Após o serverSocket ter sido criado regista-se no servidor central. Se o registo correr mal, fecha o socket aberto e encerra o funcionamento, se correr bem fica em while(true) a receber conexões.

```
while(true){
        Socket s = this.serverSocket.accept();
        System.out.println("[+] Connection from " + s.getInetAddress());
        ClientHandler c = new ClientHandler(s, users, this.clientSocket);
        new Thread(c).start();
}
```

Cria uma nova thread para cada cliente registado, cada cliente é atendido por um ClientHandler, é este que recebe os pedidos dos clientes e os processa. Podendo ser 3 pedidos diferentes, registo de cliente, pedido de consulta de música vindo de um cliente ou um pedido de consulta vindo do servidor central.

O pedido de registo verifica apenas se o utilizador já se encontra registado ou não, se não está registado cria um novo utilizador e guarda no HashMap, caso já exista envia o erro ao cliente. Também pode ser uma mensagem de registo com o código ‘o’ que serve para fazer logout (desconectar o cliente) quando o mesmo pede para sair da aplicação.

Os pedidos de consulta são semelhantes, a única diferença é que o pedido vindo de outro servidor faz *request* a todos os clientes ligados, enquanto o outro faz a todos excepto ao cliente que fez o pedido. Um pedido de consulta vindo de um cliente faz o envio do pedido ao servidor central caso nao exista a música localmente e o pedido do servidor central apenas responde com os dados da pesquisa local ao servidor central.

São enviados aos clientes ligados pedidos de consulta para uma musica requisitada, aos quais cada cliente responde se tem ou não a música disponível para envio. Com um timeout de 2 segundos, caso nao respondam assume-se que não tem a música.

```
Thread.sleep(2000);
...
```

if(inFromServer.available() == 0)

System.out.println("[+] No response from " + ipS);

No else verificamos a resposta do cliente, que pode ser um PDU com *FOUND(1)* out *NOT_FOUND(0)* , no primeiro caso são guardadas as informações do cliente para depois serem enviados ao servidor os dados dos clientes que tem a música disponível. Caso seja encontrada a música em algum dos clientes é enviado ao cliente que fez o pedido os dados dos clientes onde a música está disponível, caso nenhum cliente tenha a música é enviado um pedido ao servidor central para o mesmo fazer o pedido aos outros servidores.

● Cliente

O cliente é a unidade mais complexa do sistema, este tem que ter capacidade de fazer pedidos TCP ao seu servidor, receber pedidos TCP do seu servidor, enviar dados por UDP, receber dados por UDP.

A classe Client.java recebe dois argumentos, a porta TCP e porta UDP, por exemplo, 3001 4444. São estas as portas onde receberá pedidos TCP e datagramas de UDP, sejam pedidos ou envio/receção de dados.

Contém um *ArrayList<String> songs* que guarda as músicas disponíveis para envio, que é carregando lendo um ficheiro txt que é criado por um script em bash com as musicas disponiveis na pasta audiofiles.

songs = new ArrayList<>(availableSongs("src/cc/songs.txt"));

Depois disto conecta-se com o servidor, *Socket clientSocket = new Socket(ip, portServer);* em que o IP e porta do servidor já são conhecidos.

Pede ao utilizador que indique um ID para ser identificado na rede, envia ao servidor os seus dados, através de um PDU *REGISTER* e só avança quando o cliente estiver registado no servidor.

Após isto inicia uma *thread* para receber consultas do servidor, esta *thread* é ClientConsult que abre um *socket* TCP na porta de consulta do cliente, este apenas recebe pedidos de consulta de músicas, é esta *thread* que verifica a existência da música e caso exista informa o servidor que tem disponível enviando as informações, IP e porta UDP. E caso tenha a música disponível cria uma nova *thread* (ClientUDP.java) que fica à espera de conexões UDP, podendo receber dois tipos de pedidos, *PROBE_REQUEST* e *REQUEST* aos quais responde com timestamp e envio da música respetivamente.

Depois o cliente fica num *while* até o utilizador pedir para fazer *EXIT* , caso o utilizador faça um *CONSULT_REQUEST,* pede o nome da banda e a música a procurar e envia o pedido ao servidor, depois disto fica à espera da resposta do

servidor com um timeout de 15 segundos, se não houver resposta assume que não existe a música.

Se o servidor responder, verifica-se se foi encontrada ou não a música, caso tenha sido são feitos os seguintes passos:

- Guardar informação dos clientes que têm a música;
- Fazer *probe requests* a todos para calcular o OWD através de *timestamps,* estes pedidos são feitos via UDP;
- Guardar apenas o cliente com menor OWD;
- Requisitar música a este cliente, via UDP;
- Receber número total de segmentos da música;
- Receber o primeiro segmento;
- Verificar o número do PDU;
- While até receber PDU com número 0 (PDU final);
- Verificar quais os segmentos em falta para pedir retransmissão;
- Guardar a música.

- PDUs e formato

Os PDUs são criados no formato:
xxxxxxx|yyyyy|zzz ...

Nos quais os primeiros 7 items correspondem ao cabeçalho, o resto encontra-se separado por '|' por exemplo o PDU de registo de servidores é algo do género:

1010000|i|server|192.168.1.155|3000|

Testes e resultados

Nesta parte iremos mostrar imagens da execução da aplicação.

```
run:
[+] Server Central socket created on IP 192.168.1.155 port 4000
[+] Connection from /192.168.1.155
[+] PDU received: 1010000|i|server|192.168.1.155|3000|
[+] New server 192.168.1.155 registed
```

Figura 3. Servidor central.

```
run:
[+] Server socket created on IP 192.168.1.155 port 3000
[+] Server registed!
[+] Connection from /192.168.1.155
[+] Connection from /192.168.1.155
[+] PDU received: 1010000|i|andre|192.168.1.155|3002|
[+] New user andre created
[+] PDU received: 1010000|i|gil|192.168.1.155|3001|
[+] New user gil created
[+] PDU received: 1020000|Dzrt|000001.mp3| from user: gil
[+] Request response: 1030000|FOUND(1)|1|andre|192.168.1.155|4445|
```

Figura 4. Servidor local.

```
run:
[+] Please regist yourself
[+] User ID?
gil
[+] Sent to server
[+] User gil registed!
[+] New Request:
[+] 1. CONSULT_REQUEST
[+] 2. EXIT
1
[+] Insert band name
Dzrt
[+] Insert song name
000001.mp3
[+] Sent to server
[+] Waiting for response....[+] Song found!
[+] My timeStamp: 22-47-27.913
[+] Timestamp: 1050000|22-47-27.909|
[+] Best client: User{id=andre, ip=192.168.1.155, porta=0, portaUDP=4445} with a OWD o
[+] Receiving song 000001.mp3
<=========================50%=========================>
[+] Song received
[+] New Request:
```

Figura 5. Cliente 1 - gil.

```
run:
[+] Please regist yourself
[+] User ID?
andre
[+] Sent to server
[+] User andre registed!
[+] New Request:
[+] 1. CONSULT_REQUEST
[+] 2. EXIT
[TCP] Request from server /192.168.1.155
[TCP] PDU received: 1020000|Dzrt|000001.mp3|
[UDP] Request received: 1040000|
[UDP] Request received: 1060000|Dzrt|000001|mp3|
[UDP] Song 000001 requested.
[UDP] Song loaded, ready to send.
[UDP] Song sent
```

Figura 6. Cliente 2 - andre.

Estas imagens demonstram o funcionamento da aplicação para um servidor central, um servidor e dois clientes a correr na mesma máquina. O sistema foi também testado com duas máquinas diferentes, em várias situações, servidor central, servidor e um cliente na mesma máquina e outro cliente na outra máquina, outra situação, servidor central, servidor e cliente numa máquina e servidor e cliente noutra máquina. Ambas as situações funcionam quer na pesquisa local, quer remota, ou inter servidores. Para tal basta configurar os clientes e servidor local com os IPs a serem utilizados para garantir a correta conexão.

Conclusões e trabalho futuro

Foi um trabalho desafiador e bastante complexo que terminamos com grande satisfação e objetivos atingidos. No fim deste trabalho podemos dizer que ganhamos conhecimentos sólidos nos protocolos que utilizamos, sobre o seu funcionamento, otimização e utilização. Foi para nós mais simples utilizar o protocolo TCP do que o protocolo UDP, ao nível de programação e ao nível de compreensão de erros e otimização.

Para trabalho futuro deixamos alguma otimização à aplicação, adição de mais timeouts, melhor capacidade de tratar pedidos, etc. Uma interface mais apelativa e mais funcionalidades como listar músicas ou até reproduzi-las.