TP2: Serviço de transferência rápida e fiável de dados sobre UDP

Henrique José Carvalho Faria

José André Martins Pereira

Ricardo Leal

University of Minho, Department of Informatics,

4710-057 Braga, Portugal e-mail: {a82200,a82880}@alunos.uminho.pt

Introdução:

Na Unidade Curricular de Comunicações por Computadores, foi-nos proposta a elaboração de um serviço de transferência rápida e fiável de dados sobre uma conexão UDP genérica.

Os objetivos centrais desta aplicação são garantir inicialmente a conectividade entre cliente e servidor, autenticação e registo do cliente, listagem dos ficheiros existentes no servidor para o cliente, a transferência de dados/ficheiros entre o servidor e cliente, isto é, *download*, e por fim, todas estas funcionalidades distribuídas para vários clientes, isto é, um servidor multi-cliente.

Especificação do protocolo:

A classe **Pacote**, onde o seu nome é bastante representativo do seu prepósito, é muito importante, pois contém os vários campos necessários para o processamento dos mesmos, dos quais:

- **Id** – para ordenar pacotes
- **Type** – para diferenciar pacotes
- **Ack** – para diferenciar pacotes dentro do próprio **type**
- **NSeq** - para identificar os dados que se estão a transferir
- **Tempo** – para ser enviado para o servidor e vice-versa
- **Checksum** – para verificar integridade da mensagem
- **Data** – os dados (payload) (Máximo de 1024 bytes)

Implementação:

A arquitetura da aplicação está dividida em dois subsistemas, o dos clientes e o do servidor, tal como se pode verificar na figura acima, com a separação da linha vermelha. A aplicação é constituída por várias classes, sendo que três dessas, são muito importantes, pois, referem-se às entidades destes sistemas, que são: **ThreadCliente**, **ThreadPrincipalServidor** e **ThreadServidor**.

A **ThreadCliente**, corresponde à aplicação do lado da entidade **cliente**, a qual faz pedidos ao servidor e espera por respostas. A classe **Cliente**, é responsável por guardar o estado do mesmo, como tempos, username, password, estado de autenticação. A classe **AgenteUDP** é constituída por um *socket*, que irá ser usado para comunicar com o servidor, para enviar e receber pacotes de forma segura, a mesma também é usada no lado do servidor.

A **ThreadPrincipalServidor**, está à escuta com um *socket* genérico e único no lado do servidor, na **porta 7777**, sendo esta classe responsável por criar uma **ThreadServidor** para cada cliente, identificando-a com o **IP** e **PORTA** do mesmo, e após esta criação, trata-se de um intermediário, entre os **clientes** e as suas respetivas **ThreadServidor**, que está sempre à escuta.

Tal como já foi dito, esta thread é um intermediário, isto é, ao receber um pacote, obtém o **IP** e **PORTA** do remetente, para verificar se já existe uma **ThreadServidor** para o mesmo, e em caso afirmativo, acorda a thread em questão e envia-lhe o pacote para a mesma processar.

O grupo decidiu, que é vantajoso criar a thread imediatamente no início, no primeiro pacote do teste de conexão, pois um dos principais objetivos é ter a **ThreadPrincipalServidor**, o mais tempo possível desocupada, pois esta tem que "atender" todos os clientes, deste modo, deixa-se o trabalho de processar todos os pacotes, para a **ThreadServidor** do respetivo cliente.

O lado negativo desta implementação poderia ser, que caso o cliente não conseguisse testar a conexão, iria ficar um thread, apesar de adormecida, em memória, mas a equipa garante que caso não haja sucesso na conexão, a thread é removida, não havendo assim qualquer problema.

A **ThreadServidor**, que está associada a cada cliente, é responsável por processar os pacotes recebidos, isto é (todos estas funcionalidades serão explicadas em pormenor mais adiante): Teste conexão, registo, autenticação, listagem dos ficheiros, transferência de ficheiros(download), fim de conexão.

A classe, está ainda preparada para receber vários pacotes ao mesmo tempo e de forma desordenada, pois contém um *buffer* que ordena os mesmos por um **ID.**

Por fim, e não menos importante, existe uma classe denominada **ServerState**, que guarda informação que é partilhada pela **ThreadPrincipalServidor** e as **ThreadServidor** de cada cliente, deste modo, como é um objeto partilhado necessita de *locks*.

A informação contida nesta classe são as **ThreadServidor** de cada cliente, identificadas pelo **IP** e **PORTA** (criando para isso uma classe chamada FireTuple, que guarda os mesmos e converte-os em String)**,** dos mesmos, e o estado da conta do **Cliente**, identificado pela chave **username** do mesmo, para este se poder autenticar.

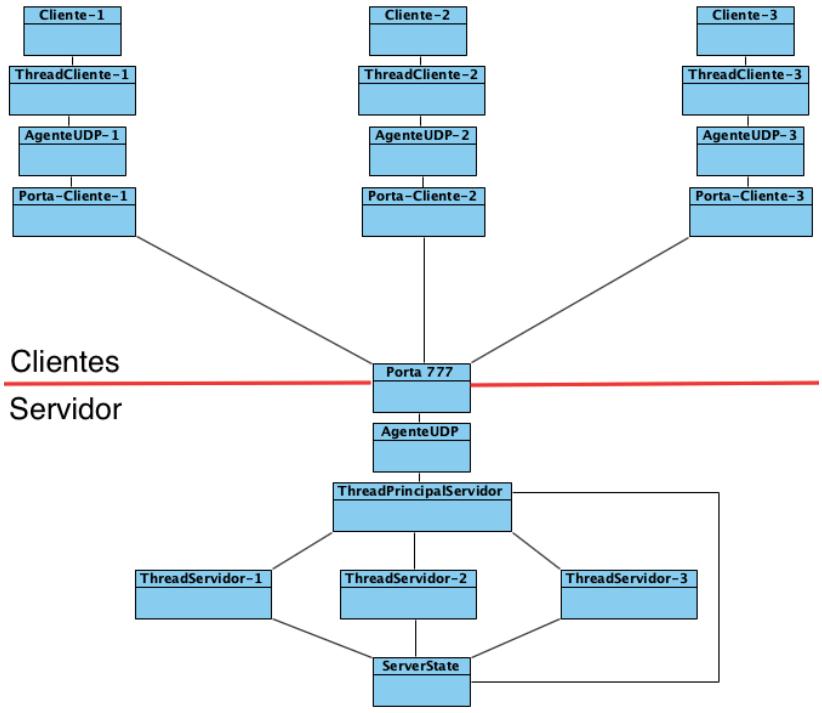

Figura 1 – Arquitetura da aplicação.

Teste de conexão:

Tal como se pode observar na imagem abaixo, o teste de conexão é feito com um **3-way handshake**, isto é, quando o **Cliente**, quer estabelecer conexão com o servidor, na **ThreadCliente** envia-se um **Pacote** com os campos **type = 'S'** e **ack = 0**, esperando um determinado tempo pela resposta.

Quando a **ThreadPrincipalServidor** recebe este tipo de pacotes, vai verificar se este cliente já tem uma **ThreadServidor** criada no **ServerState**, em caso contrário cria e envia o **Pacote** recebido para essa thread, atualizando o **ServerState** com a nova thread.

A **ThreadServidor** ao receber este **Pacote**, vai responder da mesma forma, com os mesmos campos, esperando um determinado tempo pela resposta do cliente.

A **ThreadCliente**, ao fim de esperar o tempo previsto, tal como foi dito antes, verifica se recebeu o pacote resposta do servidor e caso este seja com **type = 'S'** e **ack = 0**, o cliente vai enviar para a **ThreadPrincipalServidor**, visto que, é o intermediário, um **Pacote** com os campos, **type = 'S'** e **ack = 1**.

Do lado do servidor, quando a **ThreadPrincipalServidor** (intermediário) receber este último pacote, envia para a **ThreadServidor** destinatária do mesmo. Quando esta o receber, verifica se os campos são **type = 'S'** e **ack = 1**, e finaliza-se com sucesso o teste de conexão.

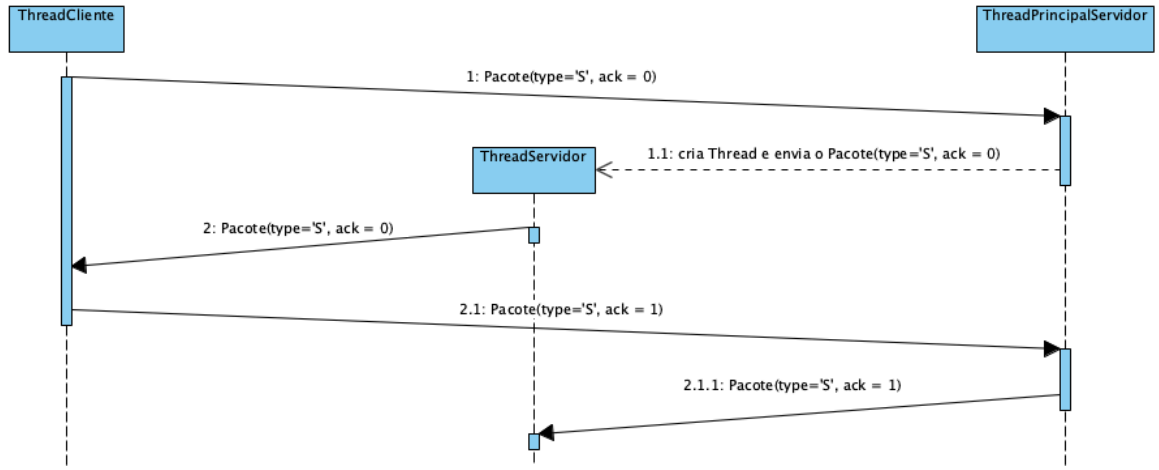

Figura 2 – Teste de conexão.

Autenticação do cliente:

O processo de autenticação do cliente, começa na **ThreadCliente**, com o envio de um **Pacote** com **type='A'** e **data = 'username;password'**.

Quando a **ThreadPrincipalServidor**, recebe o pacote, envia para a **ThreadServidor** referente a esse cliente, de seguida, essa classe, acede ao **ServerState**, tendo que fazer *lock* deste objeto, para verificar se o **username** e **password** estão corretos, existindo quatro possíveis respostas que são diferenciadas pelo valor do campo **ack** que existe no **Pacote**.

Registo do Cliente:

Do mesmo modo que a autenticação, o registo do cliente procede-se da mesma forma, sendo que a diferença rege-se pelo valor do campo **type='R'**, e a forma como o pacote é processado, pois neste caso, temos

que verificar se o **username** que o cliente escolheu já está a ser utilizado, pois este tem que ser único, devido ao facto de ser **chave**. Assim, existem duas possíveis respostas do servidor:

Transferência de Ficheiros (Download)

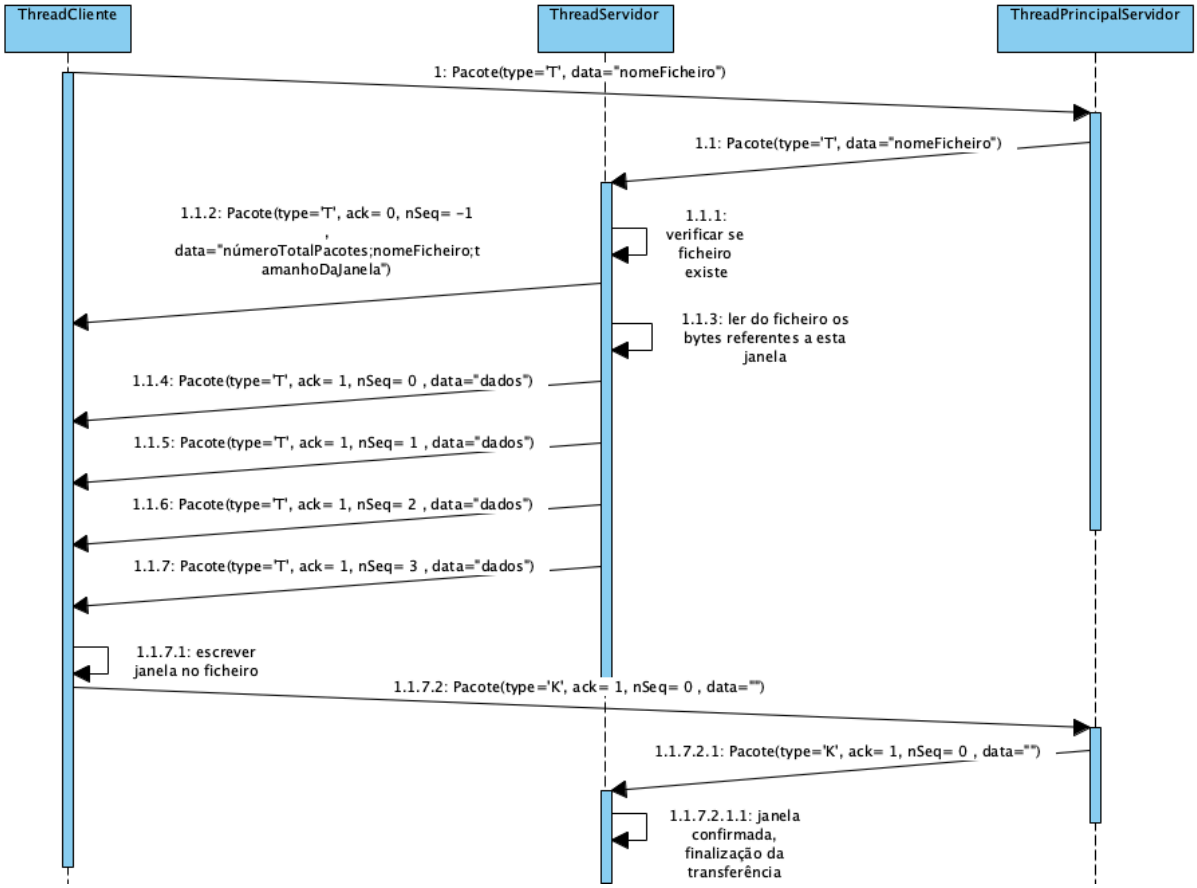

Figura 3 – Transferência de ficheiros.

A transferência de um ficheiro começa com o envio de um pacote por parte do **ThreadCliente**, com os campos **type = 'T'**, **data = 'nomeFicheiro'**.

A **ThreadServidor**, quando recebe esse pacote, inicialmente verifica se esse ficheiro existe, caso não exista, é enviado um pacote com **type = 'E'**, a informar do erro, caso contrário, calcula-se o número de pacotes que vão ser necessários para a transferência, assumindo a figura acima como exemplo, o ficheiro tem **tamanho = 4096 bytes = 4 KBytes**, visto que se tem uma **MTU = 1024 bytes**, então serão necessários **4 pacotes (4096/1024 = 4**), caso o resto da divisão **(tamanho do ficheiro % MTU != 0)**, então soma-se uma unidade ao resultado.

Após calcular o **número total de pacotes** necessários, essa informação é enviada ao cliente, bem como o, **nome do ficheiro** e o **tamanho da janela/bloco,** sendo esta informação muito importante, para que, o cliente saiba quando termina a receção de pacotes e quantos pacotes vai receber por bloco.

De seguida, envia-se a **janela/bloco** dos pacotes, isto é, **4 pacotes por bloco**, e no final de cada bloco, a **ThreadCliente**, envia um **pacote de confirmação** para o servidor, a informar se recebeu todos os pacotes e em caso contrário quais os pacotes que não recebeu, ou seja, *stop and wait*.

No exemplo da figura, pode-se verificar que foram enviados os quatro pacotes com os dados e o cliente recebeu-se todos os pacotes do ficheiro, que por acaso, apenas se necessitou de uma janela/bloco, para enviar todo o ficheiro, no entanto a aplicação está preparada para ficheiros que necessitem de mais janelas, bem como

ficheiros que não necessitem se quer de um bloco, isto é, por exemplo um ficheiro com 20 bytes, apenas vai necessitar de um pacote e tamanho de janela = 1.

O valor do número de sequência é muito útil, pois permite saber quais os dados que se tem que enviar, sendo **MTU*Número de sequência** o byte inicial e o número de bytes a ler igual ao **MTU**, havendo o caso especial, de o número de bytes a ler ser inferior à **MTU**, tendo que se fazer esse cálculo, no entanto a aplicação está preparada para esses casos.

O número de sequência também ajuda a saber se um determinado pacote recebido no cliente, está dentro do intervalo suposto, isto é, se é maior que a janela atual, e menor do que a próxima.

Importante referir que, de forma a maximizar a eficiência e face aos recursos limitados de memória **RAM**, apenas se lê para esta memória, os bytes correspondentes aquela janela/bloco, pois devido a ser um servidor multi-cliente, era bastante ineficiente e impossível ler para a memória todos os ficheiros.

Por exemplo, se tivéssemos cinquenta clientes a transferir simultaneamente um ficheiro de 100 MBytes (um ficheiro relativamente pequeno), o servidor não aguentava, pois eram aproximadamente 5GBytes em memória RAM, desta forma, permite-se uma maior capacidade nesse sentido. Do mesmo modo, também se implementou no lado do cliente a mesma estratégia, com os mesmos objetivos.

Tal como se pode verificar na figura, no fim de receber os quatro pacotes, a **ThreadCliente** envia um pacote com os campos **type = 'K'** e **data = ""**, que corresponde ao pacote de confirmação, e como o campo **data** está vazio, significa que o cliente recebeu todos os pacotes desta janela.

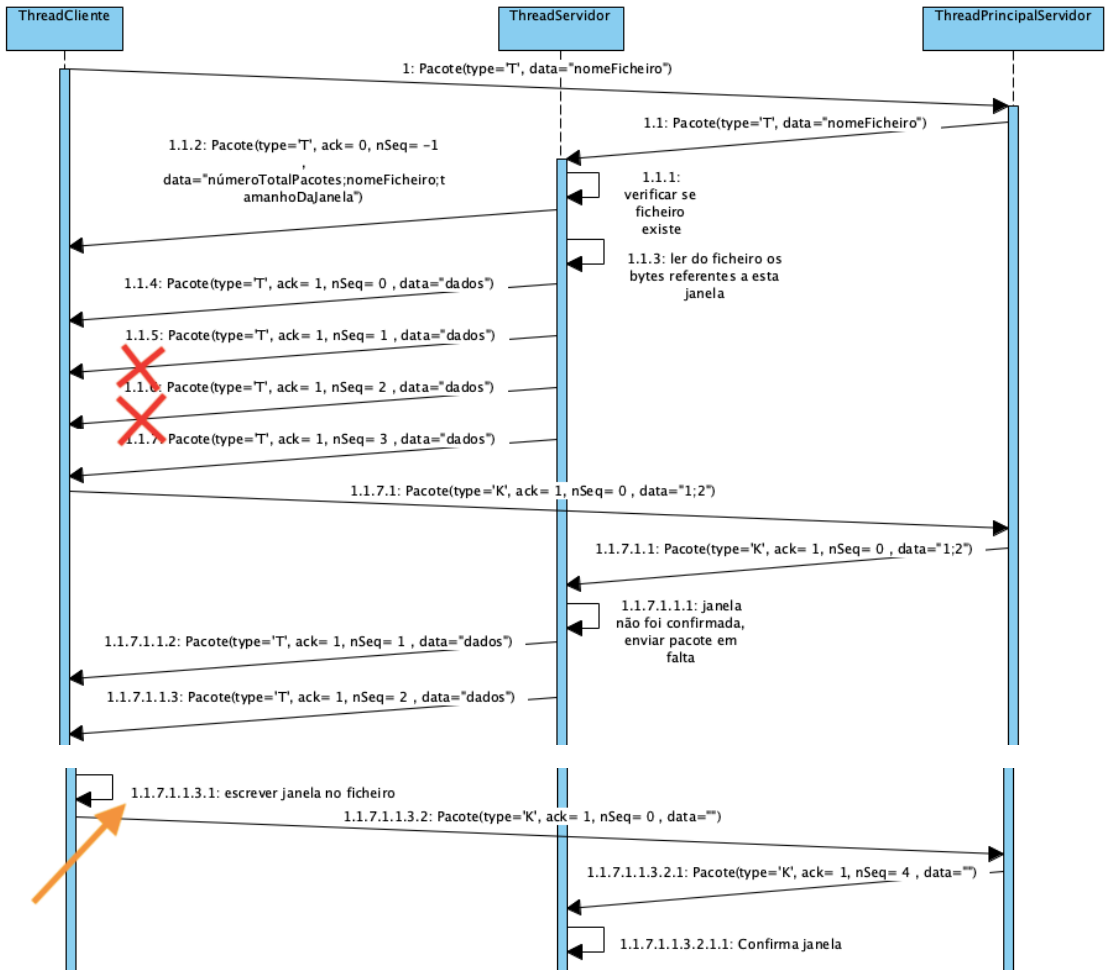

Figura 4 – Ocorrência de perdas na transferência de ficheiros.

Tal como se pode observar na figura acima, e assumindo o exemplo anterior, um ficheiro de **4 KBytes**, ou seja, **número total de pacotes = 4** e **MTU** = **1024 Bytes**. Durante o envio do primeiro janela/bloco, ocorreram duas perdas, dos pacotes com número de sequência 1, 2.

O cliente, após esperar a chegada da janela, ocorre **TIME_OUT**, isto é, não chegam os pacotes durante o prazo estipulado, chegando apenas dois que é diferente de quatro que era o esperado, e assume-se que se perderam, ou seja, ocorreram perdas.

A **ThreadCliente**, nesta situação, no pacote de confirmação, irá incluir no campo data, os números de sequência dos pacotes perdidos, separados por ";" para informar o servidor do sucedido, para que este os volte a enviar.

A determinação dos pacotes que não chegaram é feita da seguinte forma: este exemplo trata-se da primeira janela/bloco que se está a receber, ou seja, o número de sequência do primeiro pacote é zero, que é igual ao número de pacotes que eu já recebi, que também é zero, e o número de sequência do último pacote a receber é igual ao **número de sequência inicial + tamanho da janela/bloco**, sabendo que se tem que receber quatro pacotes (tamanho da janela/bloco), sabe-se os pacotes em falta.

Nos casos de perdas, também é necessário referir que a escrita no ficheiro só acontece quando toda a janela é confirmada, tal como se pode ver na figura acima, com a seta laranja, só nesse ponto é que se escreve no ficheiro a janela, pois só nesse instante é que se tem a certeza que a janela está correta e confirmada.

No caso de se perder o pacote de confirmação, o servidor, reenvia novamente toda a janela, e neste caso ocorrem duas situações possíveis. O pacote de confirmação que se perdeu, avisava que havia pacotes perdidos, mas com o reenvio de toda a janela, esses pacotes já vão ser enviados, corrigindo essa perda, e os que não tinham sido perdidos, são descartados neste segundo reenvio. O pacote de confirmação que se perdeu, tem o campo **data = vazio**, isto é, não há pacotes perdidos, logo quando o servidor reenviar toda a janela , o cliente simplesmente vai ignorar os pacotes dessa janela, visto que já tinha recebido todos os pacotes, e no próximo pacote de confirmação irá avisar que estão em falta os pacotes da próxima janela.

Por fim, e não menos importante, foi bastante útil a utilização da classe **RandomAccessFile**, que permite o posicionamento do cursor no ficheiro, visto que, se está a ler janela/bloco a janela/bloco, quando ocorrem perdas é necessário "puxar" o mesmo para trás, para se voltar a ler o ficheiro.

Integridade dos pacotes:

Para garantir que a mensagem recebida não foi alterada ou sofreu uma modificação, decidiu-se implementar um mecanismo de **"checksum"**.

Para tal, a equipa deparou-se com um dilema, maiores garantias de que a mensagem não foi alterada versus tamanho ocupado pelo **checksum** a ser enviado.Se escolher-se a primeira opção a resposta seria utilizar o **sha512**, caso contrário devia-se implementar o **sha1**. Ambos são mecanismos de **hashing** rápido para utilizar em **checksum**, a diferença está no número de combinações possíveis logo maior garantia da **integridade** da mensagem recebida.

Após longa reflexão, optou-se por aplicar **sha1** ao resultado da aplicação do **sha512** á mensagem. Desta forma garantiu-se com maior segurança não só que a mensagem não foi alterada como também se reduziu o espaço ocupado pelo checksum, no cabeça-lho do **Pacote**.

Funcionalidades adicionais:

A nível de funcionalidades adicionais às previstas pelo enunciado, tem-se a possibilidade de ter vários clientes conectados e executar ações em simultâneo. A aplicação permite, também a transferência de diversos tipos de ficheiros como vídeos (.mov), imagens (.png), ficheiros de texto (.txt), ficheiros comprimidos (.zip).

Outra funcionalidade é a possibilidade de poder transferir vários ficheiros em simultâneo, isto é, **get ficheiro1 ficheiro2 ficheiro3 ficheiro4**.

Tal como já foi dito, a característica de ler/escrever os ficheiros janela/bloco a janela/bloco, permite ter vários clientes a transferir ficheiros, sem colocar em causa a memória **RAM**, tanto do servidor como do cliente. A existência de contas para cada cliente, e a impossibilidade de se conectar em diferentes dispositivos, devido ao facto de existir estado referente à máquina em que o cliente se encontra.

Como executar a aplicação:

O grupo desenvolveu uma *makefile*, para auxiliar no momento de executar a aplicação do cliente e servidor, sendo respetivamente:

- make runServidor
- make runCliente.

Testes da aplicação:

Os testes da aplicação passaram por testar as funcionalidades do mesmo com diferentes ficheiros, tais como vídeos (.mov), imagens (.png), ficheiros de texto (.txt), entre outros.

Para garantir que as retransmissões de pacotes em casos de perdas estão funcionais, elaborou-se uma pequena condição, que através de uma classe **Random**, provoca perdas de pacotes, permitindo à equipa testar essa funcionalidade, tal como se pode verificar abaixo, esse excerto de código que está escrito no método que envia os pacotes na transferência de ficheiros.

Por fim, também se testou a mesma, com ficheiros de grandes dimensões (1.6GBytes), pelo que houve sucesso na transferência.

Conclusões:

Em suma este trabalho, ajudou a reforçar os conhecimentos adquiridos no decorrer das aulas desta disciplina, e um maior conhecimento dos protocolos **UDP** e **TCP**. Neste trabalho foi implementada uma comunicação entre um servidor e vários clientes para downloads de ficheiros de forma rápida e fiável. A comunicação foi assegurada através de **sockets UDP**, que transmitem pacotes de até 1024 bytes com um header contendo id do pacote, tipo, número de sequência, checkSum, tempo e payload.

Com o decorrer do trabalho ocorrem algumas dificuldades, nomeadamente nos tempos de espera por resposta, quer do lado do cliente, quer no servidor. Do mesmo modo, inicialmente a equipa teve problemas na transferência dos ficheiros, pois estava-se a usar para o campo data(payload) um variável de instância do tipo String ao invés de um array de bytes, pelo que a conversão de ficheiros como imagens e vídeos, não eram asseguradas.

A equipa também não conseguiu implementar uma boa versão da janela deslizante, sendo a atual implementação, envio por blocos, com confirmações ao fim dos mesmos, no entanto como não houve tempo para modificar a mesma, sugere-se para trabalho futuro essa mesma alteração. Também, foi iniciativa do grupo implementar segurança e encriptação das mensagens, no entanto,

devido à falta de tempo, para testar e garantir uma boa solução decidiu-se não implementar, considerando que se trata também de trabalho futuro.

Com esta implementação, assegurou-se não só a transmissão de pacotes, bem como se a ordenação dos mesmos, mesmo que não cheguem por ordem, integridade de cada pacote bem como retransmissão de pacotes em caso de perda ou verificação de erros no payload, através do checksum.