Relatório TP2 - PL7 Grupo 3

Comunicações por Computador

Licenciatura em Engenharia Informática

Escola de Engenharia

Universidade do Minho

2021/22

Francisco Novo A89567

Francisco Izquierdo A93241

Tiago Ribeiro A93203

Universidade do Minho
Escola de Engenharia

Índice

Introdução

No âmbito da disciplina de Comunicações por Computador e motivados pelo pretexto de que quando se trabalha em equipa, a necessidade de usar pastas partilhadas é grande, sendo um dos serviços mais básicos e indispensáveis nos projetos e no trabalho em equipa, foi-nos proposto que desenvolvêssemos um novo serviço de sincronização de pastas espontâneo e ultrarrápido, que se baseie quase exclusivamente em protocolos de transporte não orientado à conexão, como por exemplo o UDP.

Deste modo, o objetivo deste trabalho é a criação de uma aplicação denominada por *FolderFastSync*, que tem como função a sincronização rápida de pastas sem necessitar de servidores nem de conetividade Internet, baseando a funcionalidade principal da aplicação num protocolo a desenvolver de raiz para a sincronização de ficheiros sobre UDP.

Além disso, em simultâneo deve correr outro protocolo, este de monitorização simples da aplicação a desenvolver, que devolva o estado da aplicação.

O teor do trabalho recai especialmente no desenvolvimento de um protocolo capaz de lidar com as várias incoerências quando se transporta informação não orientado à conexão numa rede e uma vez que o objetivo é a sincronização de pastas, isto é, a transferência de ficheiros, qualquer informação perdida, duplicada ou até mesmo corrompida coloca em causa a integridade do ficheiro que está a ser transferido.

Posto isto, o protocolo passou pelas seguintes fases:

- Planeamento: Nesta fase o grupo planeou o teor das mensagens protocolares bem como a capacidade de resposta do protocolo face aos possíveis problemas aquando da transferência dos ficheiros.

- Implementação: Nesta fase o grupo passou do planeamento para o código, analisando e organizando o que foi planejado na fase anterior.

- Execução: Nesta fase o grupo pôs à prova a capacidade do protocolo com diversos testes primeiramente no sistema em que se encontra, mas acima de tudo testar na topologia *Core*, uma vez que esta topologia simula o “mundo real”, conseguindo assim o grupo verificar a integridade do seu protocolo nas situações adversas.

Alado ao supramencionado é fulcral escolher uma boa linguagem de programação dado o teor e rigor do trabalho proposto. Assim, o grupo decidiu realizar o trabalho com base na linguagem de programação orientada aos objetos, Java, dada a experiência que já tínhamos e pela grande disponibilidade de bibliotecas e completude que a linguagem apresenta.

Arquitetura

De forma a desenvolver o protocolo, foi necessário definir como cada etapa se ia conciliar, bem como se interligar e para isto tivemos de projetar um modelo que iria representar a arquitetura do projeto. Assim, foi concebido um modelo conceptual de forma pensada e estruturada no qual cada componente representada é relevante para o funcionamento do programa.

A arquitetura pensada e desenvolvida é composta por: **FFSync, Sender, Receiver, ReceiverWorker, HTTPServer** e **ClienteHTTP**, no qual cada entidade tem uma função especifica e em correlação com as restantes.

- **FFSync (Cliente)**: Esta é a entidade principal do programa uma vez que é esta que será posta em execução e serve de "motor" de arranque do mesmo, sendo responsável por dar início ao processo protocolar.

- **Sender**: É responsável por enviar os pacotes com as informações relevantes ao ficheiro, mas também alguns pacotes de controlo de fluxo. Por cada ficheiro que deve ser enviado, é criado uma nova entidade, de forma a possibilitar o envio de vários ficheiros em simultâneo.

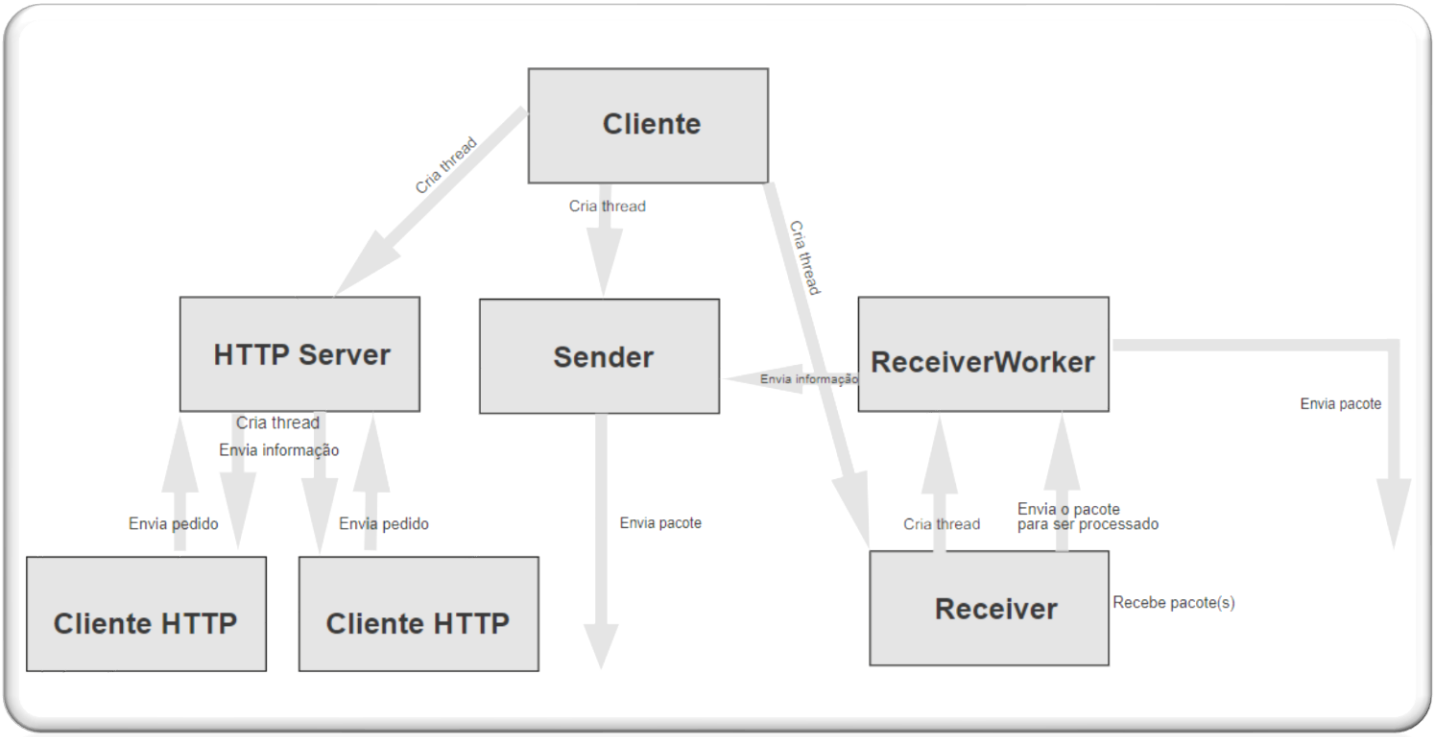
- **Receiver**: Entidade encarregue de receber todos os pacotes que são endereçados à entidade **Cliente** supramencionada, lidando por isso com todos os pacotes recebidos na porta 80 da *Socket* da entidade previamente mencionada. De forma a processar o conteúdo de cada pacote recebido, cria uma nova entidade, **ReceiverWorker**, sendo esta a responsável pelo processamento de cada pacote.

- **ReceiverWorker**: Entidade responsável pelo processamento de cada pacote recebido pela entidade acima caracterizada, avaliando o tipo e também o conteúdo de cada pacote, sendo ainda responsável pelo gestão e funcionamento de estruturas de controlo, permitindo assim sincronização entre as várias entidades e o fluxo de informação de forma controlada.

- **HTTPServer**: Entidade responsável por atender pedidos *HTTP* na porta 80, pedidos estes relacionados com o funcionamento do programa.

- **ClienteHTTP**: Entidade externa ao programa, representando qualquer cliente que faça um pedido *HTTP* acerca do funcionamento do programa.

É de realçar que a forma de comunicação entre dois clientes é feita única e exclusivamente através de uma só *socket*, isto é, cada cliente cria uma única *socket* na porta especificada, recebendo e enviando informação através desta.



1: Arquitetura da aplicação

Especificação do Protocolo

O próximo passo foi planear o funcionamento do programa e assim, tivemos de pensar e criar o protocolo no qual o mesmo iria recair, dado que o protocolo a definir teria de alcançar o objetivo principal, o de sincronizar as pastas previamente selecionadas, mas acima de tudo teria de ser pensado de forma a contornar possíveis incoerências aquando do objetivo supramencionado, tais como perda de pacotes e duplicação dos mesmos.

Com isto, separámos o protocolo em duas vertentes, **envio de informação** e **controlo de fluxo de informação**, para as quais especificámos como seriam representadas as mensagens protocolares (para efeitos de simplificação iremos denominar as mensagens protocolares como *pacotes*), uma vez que são estas o "meio de transporte" responsável pela troca de informação entre duas ou mais conexões.

Primeiramente, definimos o envio/ controlo de fluxo de informação como sendo sequencial, ou seja, é enviado e recebido um pacote de cada vez.

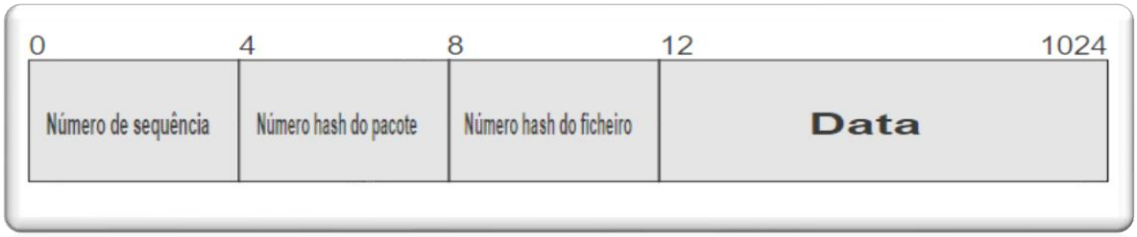
Formato das mensagens protocolares

Todos os pacotes possuem um cabeçalho de forma a haver um controlo e integridade dos mesmos, sendo que para o protocolo implementado existem cinco tipos diferentes de mensagens. Estas partilham características em comum (campos do cabeçalho) tais como:

- **Número de sequência:** Permite numerar o pacote, controlando possíveis duplicações do pacote, uma vez que é sempre guardado o número do próximo pacote que deve chegar.
- **Número *hash* do pacote:** Permite verificar a integridade da informação contida no pacote (caso a informação a analisar esteja corrompida) ou caracterizar um pacote em específico.
- **Número *hash* do ficheiro:** Permite associar o pacote recebido ao ficheiro correspondente que está a ser recebido.

Envio de informação

No envio de informação, são enviados pacotes que contêm a informação respetiva ao ficheiro que está a ser enviado, sendo estes recebidos pelo cliente que está a receber o ficheiro, construindo-o sequencialmente. Cada pacote no envio de informação tem o tamanho de 1024 *bytes*, sendo 12 *bytes* para o cabeçalho e o restante contento informação sobre o ficheiro que está a ser enviado.

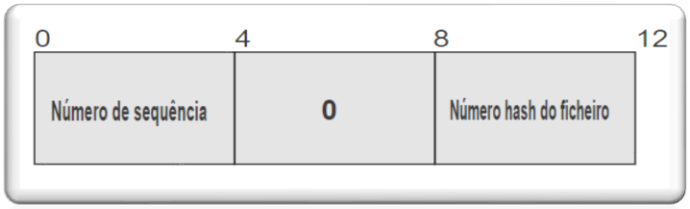


2: Pacote do ficheiro

Controlo de fluxo de informação

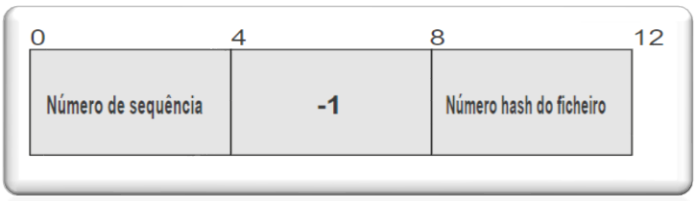
No controlo de fluxo de informação, são enviados determinados pacotes com um tamanho de 12 ou 16-1024 *bytes*, dependendo do contexto em que se inserem. Estes pacotes são caracterizados pelo campo do cabeçalho supramencionado "**número hash do pacote**", tendo funções próprias para controlar o fluxo de informação e manter a coerência da transferência do ficheiro. Estes pacotes podem ser:

- **Pacote ACK**: Pacote de 12 bytes que permite garantir que pacote do ficheiro foi recebido, está coerente e na ordem correta. Estes tipos de pacotes são identificados pelo número 0 no campo supramencionado.



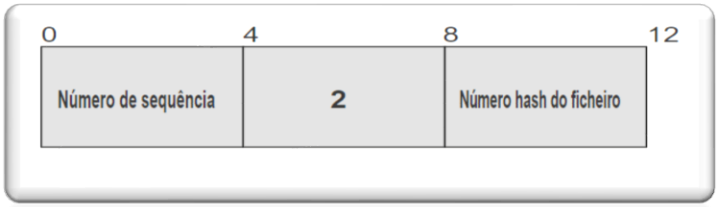
3: Pacote ACK

- **Pacote FYN**: Pacote de 12 bytes que permite garantir que toda a informação a ser enviada foi recebida, possibilitando o envio do próximo conjunto de informação ou mesmo o fecho da conexão. Estes tipos de pacotes são identificados pelo número -1 no campo supramencionado.



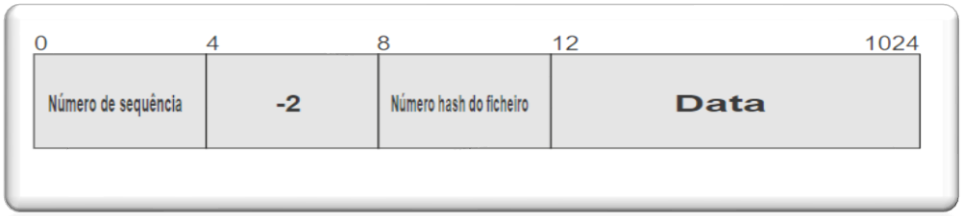
4: Pacote FYN

- **Pacote LAST**: Pacote de 12 bytes que é enviado quando todos os pacotes do ficheiro já tiverem sido enviados, de forma a determinar o fim da transferência do ficheiro. Estes tipos de pacotes são identificados pelo número 2 no campo supramencionado.



5: Pacote LAST

- **Pacote LIST**: Pacote de 16 a 1024 bytes que serve de pedido de conexão e também contém a informação sobre a lista de todos os ficheiros contidos na pasta a sincronizar, sendo enviado ao cliente que ao receber o pacote compara com a lista de todos os ficheiros da sua pasta de forma a não criar incoerências. Estes tipos de pacotes são identificados pelo número -2 no campo supramencionado.

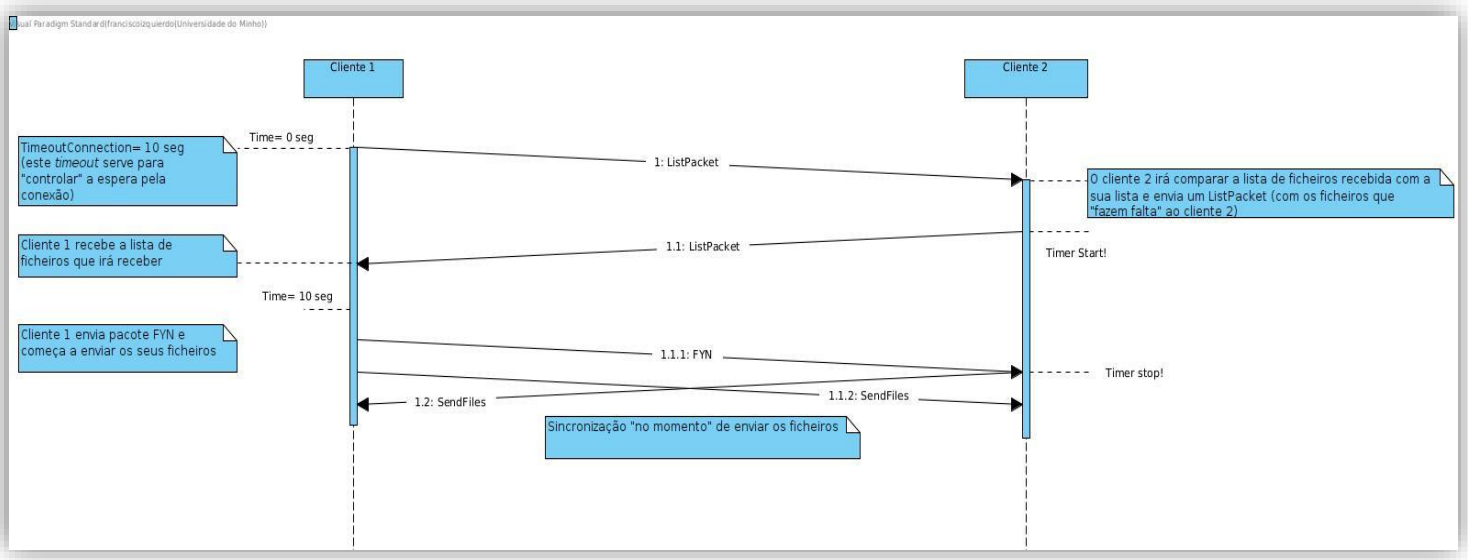


6: Pacote LIST

Interações

Após definido o teor das mensagens protocolares, o protocolo deve ainda definir especificações para o uso das mesmas em situações especificas na sincronização do(s) ficheiro(s), de forma a garantir a coerência e evitar incongruências durante a transferência dos mesmos. Assim, o protocolo especifica essas mesmas situações e como contorná-las, sendo estas situações as seguintes:

- **Início de conexão**: No início de conexão é enviado um pacote LIST da pasta a sincronizar ao cliente de destino, sendo que este analisa e compara com a sua pasta a sincronizar a lista de ficheiros, devolvendo como resposta um pacote LIST com a sua lista de ficheiros que fazem falta ao cliente remissivo. Após haver a sincronização dos ficheiros a enviar/receber, dá-se início à transferência dos ficheiros. É de realçar que é retransmitido várias vezes este pacote até obter uma resposta de volta, de forma a garantir que ambos os clientes estão conectados.



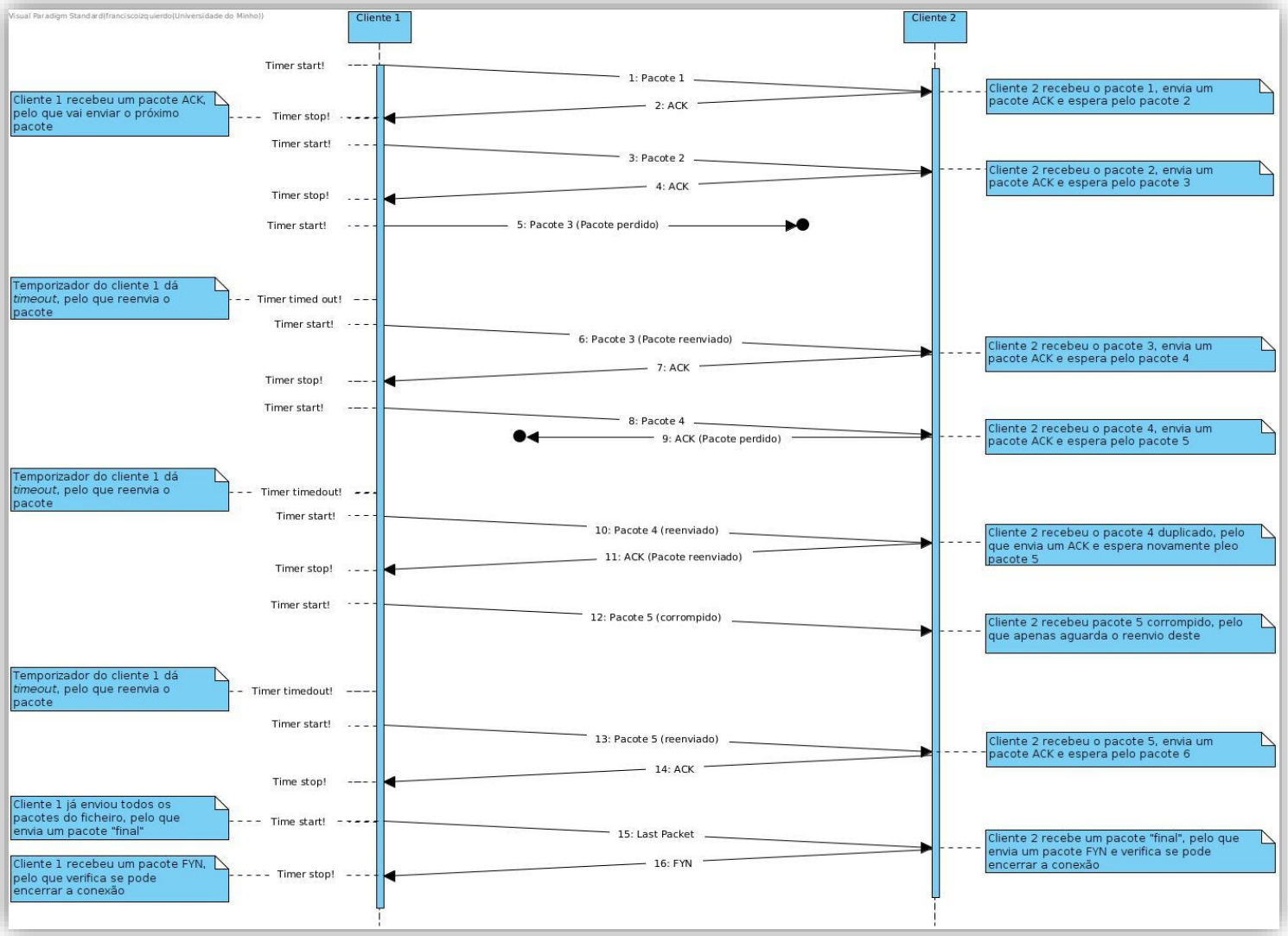
7: Início de conexão

- **Pacote de ficheiro**: Caso não ocorra incoerências no envio ou receção de informação relativa a um ficheiro que está a ser transferido, o pacote de ficheiro é recebido pelo cliente destino e este analisa-o, junta-o ao ficheiro a ser construído/sincronizado e por fim dá a conhecer através do envio de um **pacote *ACK*** ao cliente de origem que recebeu o respetivo pacote e que a informação estava concisa e integra.

- **Pacote de ficheiro perdido/corrompido**: Caso haja a perda/corrupção de um pacote correspondente ao ficheiro no envio do mesmo, o cliente que está a receber os pacotes desse ficheiro, não gera a resposta a este, esperando pelo reenvio do mesmo pacote. Esta reação é despoletada pelo "alarme" do temporizador de quem está a enviar, isto é, se não obteve a confirmação de que o pacote enviado foi recebido num intervalo de tempo (200 milissegundos) volta a reenviar o mesmo pacote.

- **Pacote de ficheiro duplicado**: Caso haja a duplicação de um pacote correspondente ao ficheiro no envio do mesmo, o cliente que está a receber os pacotes desse ficheiro, gera a resposta a este e envia-a, mas descarta a informação contida no pacote uma vez que esta é repetida.

- **Fim de conexão**: Quando toda a informação do ficheiro foi enviada, é enviado um pacote LAST sendo recebido e devolvendo como resposta um pacote FYN, de forma a garantir que foi reconhecido que já não há mais informação a receber relativa ao ficheiro. Após isto, é analisada as condições de encerramento do programa, isto é, se todos os ficheiros a serem recebidos foram enviados e vice-versa.



8: Diagrama de sequência de envio de pacotes

Implementação

Com tudo o que já foi mencionado, falta agora referir aspetos relevantes acerca da implementação da aplicação, como métodos importantes dados a sua especificação e também classes pelo mesmo motivo.

- **Métodos *run***: De forma a obter paralelismo e conseguir atender múltiplos pedidos em simultâneo nos dois sentidos de forma a tornar a sincronização mais eficiente, são usadas *threads* relacionadas às entidades descritas na arquitetura da aplicação. Assim destacam-se os métodos ***run()*** que dão início à execução da *thread* especificada.
- **Método *hash***: Todos os números *hash* supramencionados, sejam relativos ao pacote, à informação do pacote ou até mesmo ao conteúdo do ficheiro a sincronizar, provêm deste método que dado um determinado *input* gera o número *hash* associado.
- **Método *encrypt* e *getDecryptedInfo***: Todos os pacotes são encriptados, fazendo uso da palavra-passe "partilhada" por ambos os clientes e também o "**número hash do pacote**" de forma a gerar uma encriptação única e diferente para cada pacote e estes serem reconhecidos pelo cliente de destino que foi realmente o cliente de origem que os enviou. Assim, o método *encrypt()* é responsável por encriptar cada pacote a ser enviado e o método *getDecryptedInfo()* o responsável pelo processo inverso.

  Nota: Aquando da sincronização das pastas é usada uma palavra-passe, que deve ser colocada no início da conexão por ambos os clientes e que deve ser comum a ambos, isto é, segredo partilhado. Caso as palavras-passes não sejam comuns, o primeiro pacote de ambos os lados dos clientes em causa, será retransmitido várias vezes (uma vez que após o processo de desencriptação a informação contida no pacote é ilegível), pois a integridade da informação está posta em causa, encerrando a conexão com a informação de que as palavras-passes não são iguais.
- **Métodos handlers**: Cada pacote recebido pelo cliente destino, após ser identificado o teor do mesmo, é tratado pelo seu método específico, isto é, por exemplo caso seja recebido um pacote *ACK*, este será tratado pelo *handler* correspondente a este. Assim, cada pacote possui um *handler* próprio e específico para o tratamento de informação recebida.

Detalhes

Estruturas de controlo

De forma a garantir a transferência de vários ficheiros em simultâneo, além de ser usado o campo "**número hash do ficheiro**", uma vez que só é usado uma *socket* (de ambos os lados) como meio de comunicação entre os dois clientes, são usadas algumas estruturas de controlo, de forma a garantir a coerência e a correta correspondência de informação dos vários ficheiros. Assim, destacam-se 3 estruturas de controlos que são 3 *Hash Maps* que controlam o envio, a receção e a construção dos vários ficheiros. Com isto temos:

- **Hash Map Sender**: Esta estrutura de controlo faz a gestão de todos os pacotes dos ficheiros que estão a ser enviados, sendo a chave o *hash* do ficheiro e o valor o número do pacote a ser enviado. Com isto, sempre que um pacote ACK é recebido relativamente a um dado ficheiro, a entidade ReceiverWorker altera na chave do ficheiro correspondente o valor para o próximo pacote a ser enviado, de forma a prosseguir a transferência.
  Além disso, sempre que um pacote FYN é recebido, o valor associado à chave é alterado para -1, de forma a representar que a transferência do ficheiro em causa está finalizada.

- **Hash Map Receiver**: Esta estrutura de controlo faz a gestão de todos os pacotes dos ficheiros que estão a ser recebidos, sendo a chave o *hash* do ficheiro e o valor o número do pacote a ser recebido. Com isto, sempre que um pacote de ficheiro é recebido relativamente a um dado ficheiro, a entidade ReceiverWorker compara o número de sequência deste ao número do pacote de que está à espera de receber para esse ficheiro (valor relativo à chave que é a *hash* do ficheiro em causa), de forma a contornar o problema de duplicação de pacotes.
  Além disso, sempre que um pacote LAST é recebido, o valor associado à chave é alterado para -1, de forma a representar que a transferência do ficheiro em causa está finalizada.

- **Hash Map FilesReceived**: Esta estrutura de controlo faz a gestão da construção dos ficheiros que estão a serem recebidos, tendo como chave o número *hash* associado ao ficheiro e como valor o *FileOutputStream*, que é aberto sempre que um pacote de ficheiro é recebido, verificada a sua integridade e coerência, sendo a sua informação escrita para o ficheiro.

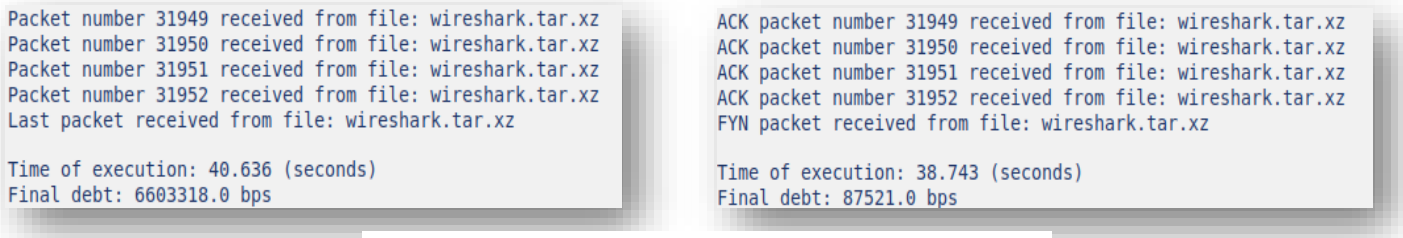
  Nota: No início de conexão, ao trocar-se a listas de ficheiros das pastas a sincronizar são comparados os ficheiros comuns e não comuns. Os ficheiros que não são comuns são recebidos pelo cliente a quem fazem falta em ambos os lados. No caso de haver ficheiros comuns, isto é, com o mesmo nome, é analisado o conteúdo dos ficheiros comuns em causa de ambos os lados dos clientes, através da geração de um número *hash* e caso este seja diferente para os dois clientes (a informação dos ficheiros com o mesmo nome é diferente), é comparada as datas da última alteração do ficheiro em causa, sendo que o cliente que possui a data mais recente da última alteração envia-o ao outro cliente. Toda a informação relativa aos números *hash* do conteúdo bem como as datas da última modificação, são geradas no arranque da aplicação.

Testes e resultados

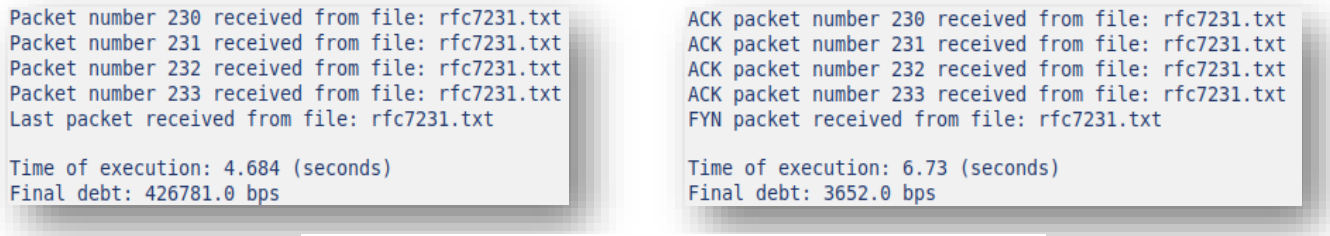
De forma a testar a eficiência do protocolo desenvolvido, o grupo realizou vários testes em diversos cenários. É de salientar que o protocolo desenvolvimento, tal como foi previamente mencionado, transporta pacotes sequenciais, isto é para cada ficheiro apenas é enviado um pacote de cada vez, mas apesar disso, o protocolo permite o paralelismo dos vários ficheiros, ou seja, o envio de vários ficheiros em simultâneo.

Para os vários cenários de teste, foi utilizada a topologia *Core* disponibilizada pelos docentes da disciplina, sendo os testes efetuados em ligações com "perturbações", de forma a pôr à prova a capacidade do protocolo desenvolvido.

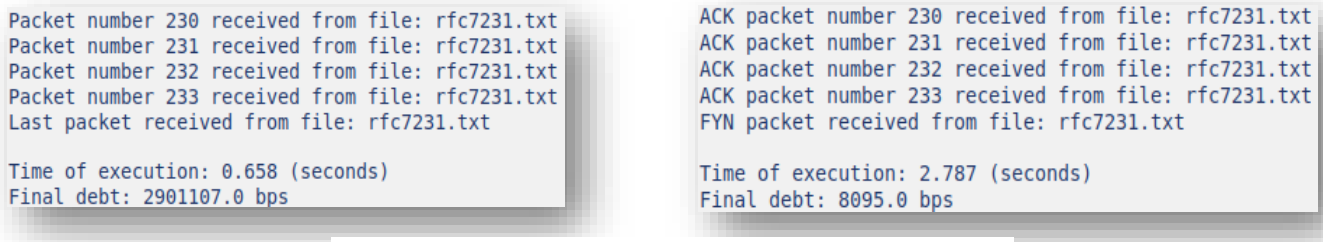
É de salientar que as imagens abaixo são referentes à informação final dos ficheiros *logs* gerado aquando da transferência de cada cenário.



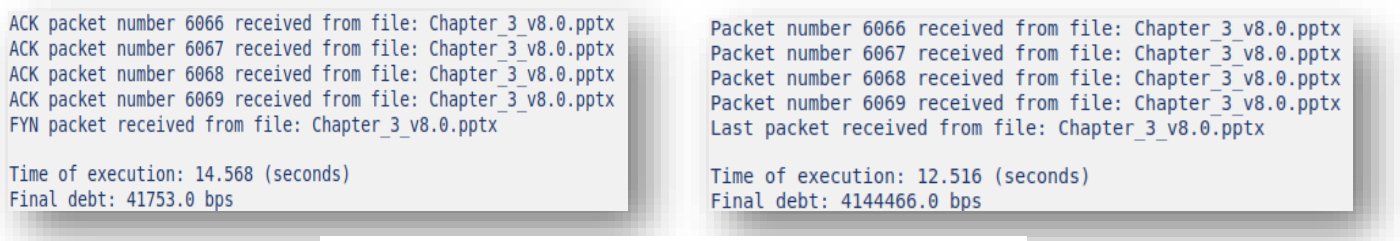
Sincronização entre pastas de 35MB (recebida, informação da imagem esquerda) e 5MB (recebida, informação da imagem direita)



Sincronização entre pastas de 235KB (recebida, informação da imagem esquerda) e vazia (recebida, informação da imagem direita) com duplicação/perda de pacotes



Sincronização entre as mesmas pastas mencionadas anteriormente: pasta de 235KB (recebida, informação da imagem esquerda) e pasta vazia (recebida, informação da imagem direita) sem duplicação/perda de pacotes



Sincronização entre pastas vazia (recebida, informação da imagem esquerda) e de 5MB (recebida, informação da imagem direita) com duplicação/perda de pacotes

Com os testes e resultados obtidos, é evidente a diferença no tempo e débito de informação quando comparado ligações “perfeitas” e ligações que introduzem erros, nomeadamente pacotes perdidos e duplicados. Assim conclui-se que na ligação com introdução de erros, o tempo de transferência bem como o débito de informação aumenta em comparação com ligações “perfeitas”, uma vez que o protocolo tem de gerar mais informação e por isso demora mais tempo para corrigir as incoerências aquando da transferência.

Conclusões e trabalho futuro

Em suma, este trabalho mostrou ser de enorme importância dado que deu a conhecer ao grupo a problemática da criação de protocolos de forma ordenada, concisa e bem pensada. O trabalho mostrou ser muito desafiante principalmente no planeamento do protocolo a criar, dado que este teria de ser posto à prova perante diversas situações do mundo real aquando do transporte de informação não orientado à conexão numa rede. Além disso, o trabalho permitiu perceber as diferenças entre transporte orientado e não orientado à conexão, as vantagens e desvantagens entre ambos em diversas vertentes, sendo que destas vertentes destacam-se a confiabilidade e rapidez do transporte de informação. Assim, concluímos que o transporte não orientado à conexão é de facto muito mais eficiente no que concerne à rapidez de informação, mas menos confiável uma vez que traz muitos mais problemas no transporte da mesma, isto é, para o teor do qual o transporte não orientado à conexão foi utilizado neste trabalho.

Com isto, percebemos que de facto o transporte não orientado à conexão é extremamente útil para por exemplo serviços de *streaming*, uma vez que os problemas abordados neste trabalho como duplicação de pacotes, perda de pacotes, é facilmente contornado enquanto na sincronização (que acaba por ser de facto transferência) de ficheiros é preciso ter-se o cuidado extra, dado que qualquer informação errática põe em causa a integridade do ficheiro.

Para concluir, este trabalho ajudou-nos a perceber também a importância de no futuro planear qualquer projeto de forma bem pensada e estruturada, de forma a contornar possíveis dificuldades, pois foi algo que se mostrou bastante relevante no desenvolvimento do trabalho. Assim, em trabalhos futuros relativos à área abordada, o transporte de informação, teremos em consideração uma boa planificação e modulação do projeto a desenvolver.