Trabalho Prático Nº2 - Serviço de Transferência rápida e fiável de dados sobre UDP

Luís Braga, Luís Martins e João Nunes

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal

{a82088, a82298, a82300}@alunos.uminho.pt

Resumo O presente relatório foi elaborado no âmbito da Unidade Curricular de Comunicações por Computador do 3º ano do Mestrado Integrado em Engenharia Informática. Este apresenta no planeamento e implementação de um serviço de transferência de dados sobre UDP.

Keywords: UDP · Transferência · Comunicação · Socket · Protocolo.

1 Introdução

No âmbito da unidade curricular de Comunicações por Computador foi proposto, no segundo trabalho prático, que se implementasse um serviço de transferência de dados, fiável, sobre uma conexão UDP genérica. O mecanismo deve ser capaz de realizar upload e download e deve ser composto, também, por uma fase de conexão (fase inicial) e outra de término de ligação (fase final). A transferência deve ser fiável e, para o efeito, deverá ter um método de verificação de integridade e outro de confirmação e retransmissão.

Para elaborar tal sistema foram elaborados um **AgenteUDP** que comunica através de datagramas UDP; outros componentes que implementam todo o transporte fiável, como por exemplo, o **TransfereCC** e outras threads que tratam da transferência e uma aplicação simples de linha de comando que permite escrever comandos do tipo GET file ou PUT file (**TransfereCC CmdLine App**).

2 Capacidades do Sistema

→ Controlo de conexão (partes envolvidas aceitam/terminam conexões).
→ Cálculo do *Round Trip Time* durante a fase de conexão para o efeito da definição do *timeout*.
→ Sincronização dos números de sequência e números de *acknowledgement* durante a fase de conexão.
→ Cálculo do *checksum* no envio e na recepção, com verificação da igualdade de ambos para o efeito de verificação de integridade e possível reenvio.
→ Reenvio se o *checksum* estiver errado.
→ Capacidade de recepção/tratamento de múltiplos pedidos.
→ Impossibilidade de realizar um *put* se o ficheiro já existir no destino, mostrando uma aviso de "*Ficheiro já existe.*".
→ Impossibilidade de realizar um *get* se o ficheiro já existir na própria máquina, mostrando uma mensagem de "*Ficheiro já existe.*". No entanto, neste caso, existe a possibilidade de lhe atribuir um nome local, tendo o comando o formato "*ccget -name <nomeLocal> <nomeNaMáquinaRemota> <IpDestino>*".
→ Implementado segundo um algoritmo Stop-and-Wait.

3 Especificação do protocolo

3.1 Formato das mensagens protocolares

Para a realização da comunicação entre **AgentesUDP** é necessáro criar um protocolo de modo a algoritmizar a comunicação.

Em primeiro lugar, é necessário determinar o Protocol Data Unit (PDU), tendo em conta as funcionalidades plausíveis de serem implementadas. Para o efeito, foi pensado o seguinte PDU:

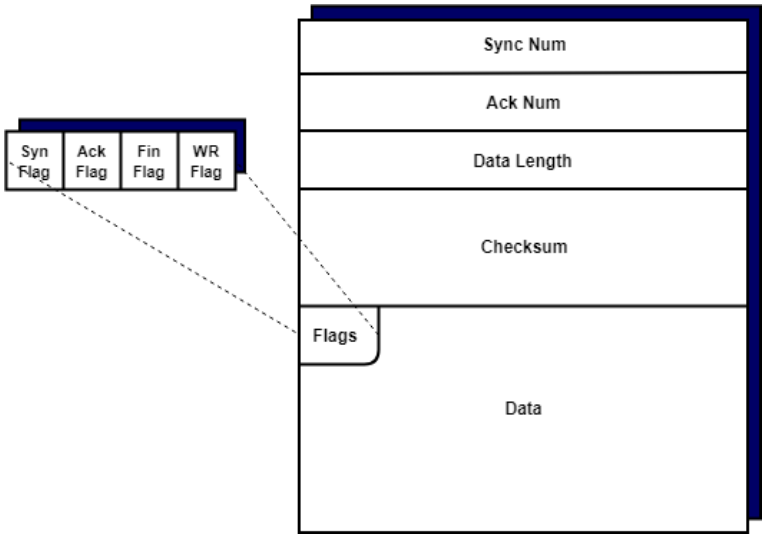

Figura 1. Formato do PDU.

É de notar que, o PDU foi elaborado tendo sempre em mente os campos dos protocolos de transporte, tanto o *TCP* como o *UDP*, entre outros, bem como os próprios campos dos protocolos de aplicação estudados no primeiro trabalho prático.

Procedendo à explicação de cada um dos campos do *PDU* elaborado:

- **Sequence Number (Sync Num):** O *Sequence Number* poderá funcionar de duas maneiras diferentes consoante o bit que consta na flag *SYN*, caso este esteja a 1, então o *Sequence Number* corresponde ao número de sequência inicial. Se a flag estiver com o bit a 0 então o *Sequence Number* que consta é o número de sequência acumulado.
- **Acknowledgement Number (Ack Num) :** Se a Ack Flag se encontrar a 1 então o valor deste campo é o próximo número de sequência do destinatário do Ack. Este campo serve para reconhecer a chegada de todos os bytes anteriormente enviados, sendo que inicialmente é apenas reconhecido o número de sequência de ambas as partes.
- **Data Length:** Campo que dá a conhecer ao destinatário o número de bytes do campo de dados.
- **Checksum:** Despois de calculado, neste campo é colocado o *checksum* do dados carregados no pacote. À chegada é calculado novamente e comparado com o que foi enviado neste campo. Em caso de erro é enviado um Ack duplicado que resultado no reenvio da mensagem.
- **Flags:** As flags são utilizadas como controlo, ocupando para tal 9 bits, existindo 9 flags no total, nas quais se destacam o *ACK*, o *FIN* e o *SYN*, que serão importantíssimos no que toca a conecção entre os dois agentes.
  - **SYN:** Assinala se a conexão se encontra numa fase inicial de sincronização.
  - **ACK:** Se no campo *ACK* a flag esteja a 1 então o valor deste campo é o próximo número de sequência que a entidade *TCP* espera receber.
  - **FIN:** Indica se alguma das partes envolventes pretende terminar a conexão. A recepção de um pacote com esta flag resulta no término da ligação.
  - **WR (Write):** Se esta flag estiver a 1 significa que se pretende realizar um put (write no destino), caso contrário é um get.
- **Data:** Neste campo transportados os dados propriamente ditos.

3.2 Interacções

Fase de Sincronização Esta é a fase inicial da conexão. É nela que os números de reconhecimento e sequência se sincronizam. Na figura que se segue podemos verificar o que acontece no decorrer do programa relativamente a esta fase.

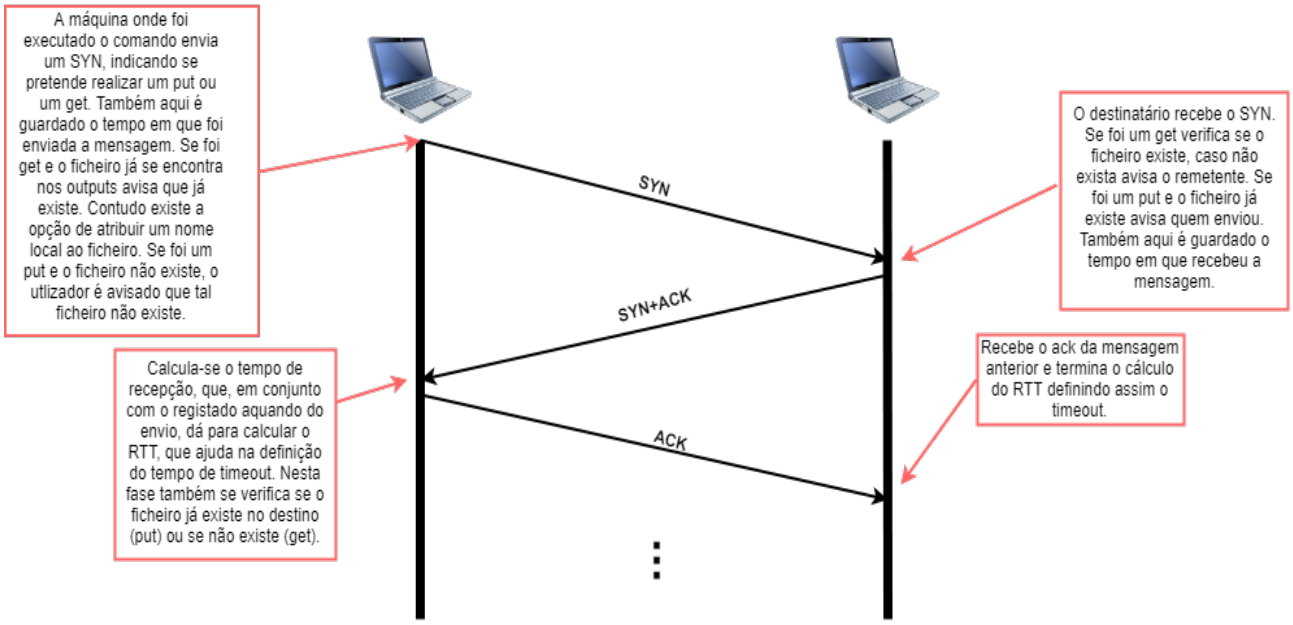

Figura 2. Diagrama representativo da fase de conexão.

Para além das informações dadas no diagrama, e mais uma vez, também é nesta fase que os números de sequência e reconhecimento são sincronizados

Fase de Transporte de dados Nesta fase decorre o transporte de dados. Esta fase é bastante simples como se pode confirmar na seguinte figura.

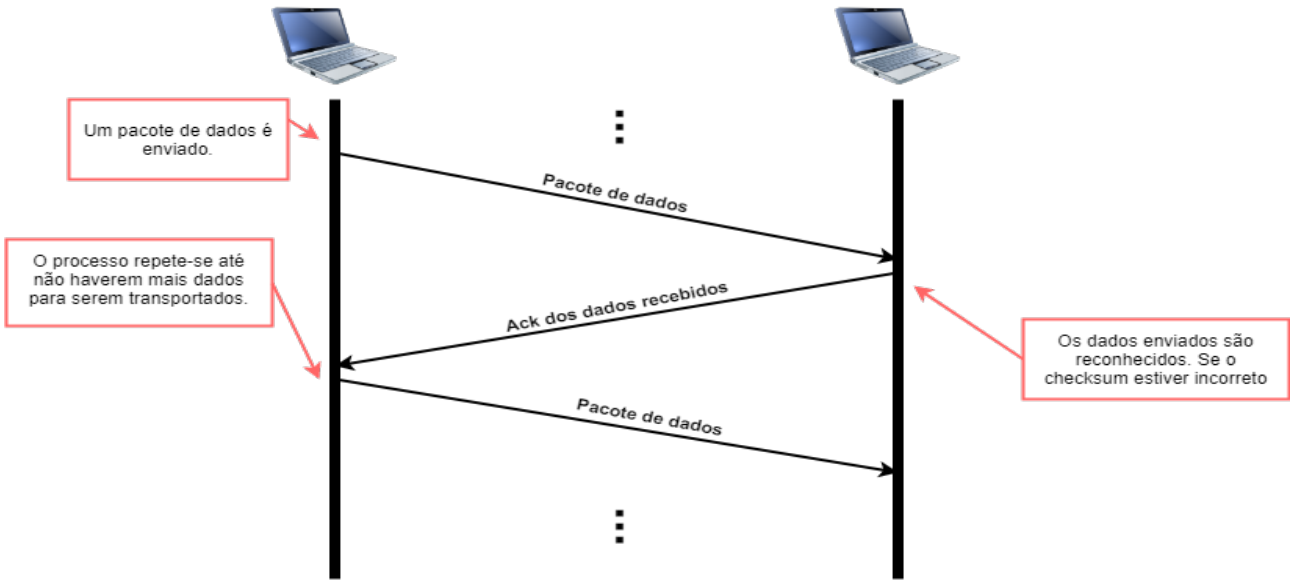

Figura 3. Diagrama representativo da fase de transferência de dados.

Fase de Término de ligação Nesta acontece o término de ligação que é lançado por quem envia os dados.

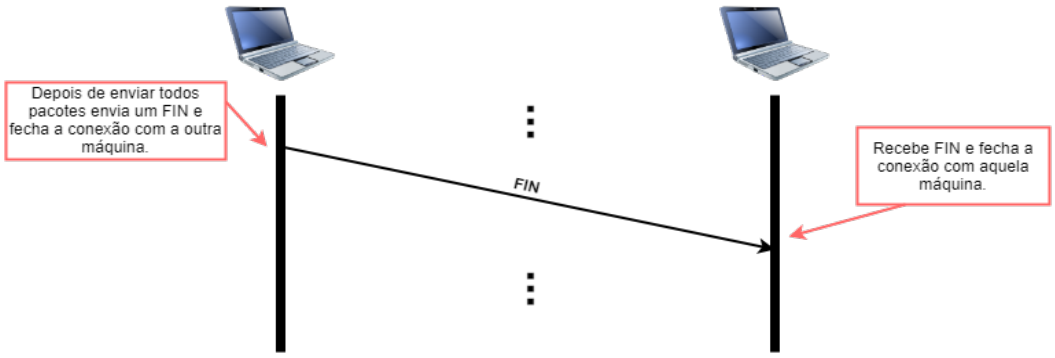

Figura 4. Diagrama representativo da fase de desconexão.

Resumo do protocolo Com a imagem apresentada de seguida podemos ter uma visão geral do protocolo utilizado.

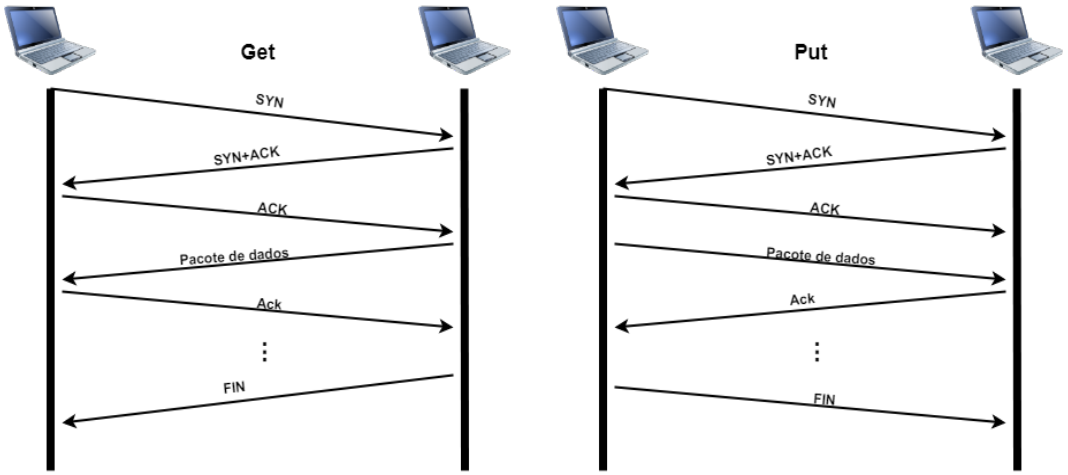

Figura 5. Diagrama representativo dos possíveis comandos.

4 Implementação

4.1 Arquitectura da solução

Com a finalidade de resolver o problema proposto implementou-se o sistema cujo funcionamento geral pode ser visto de seguida.

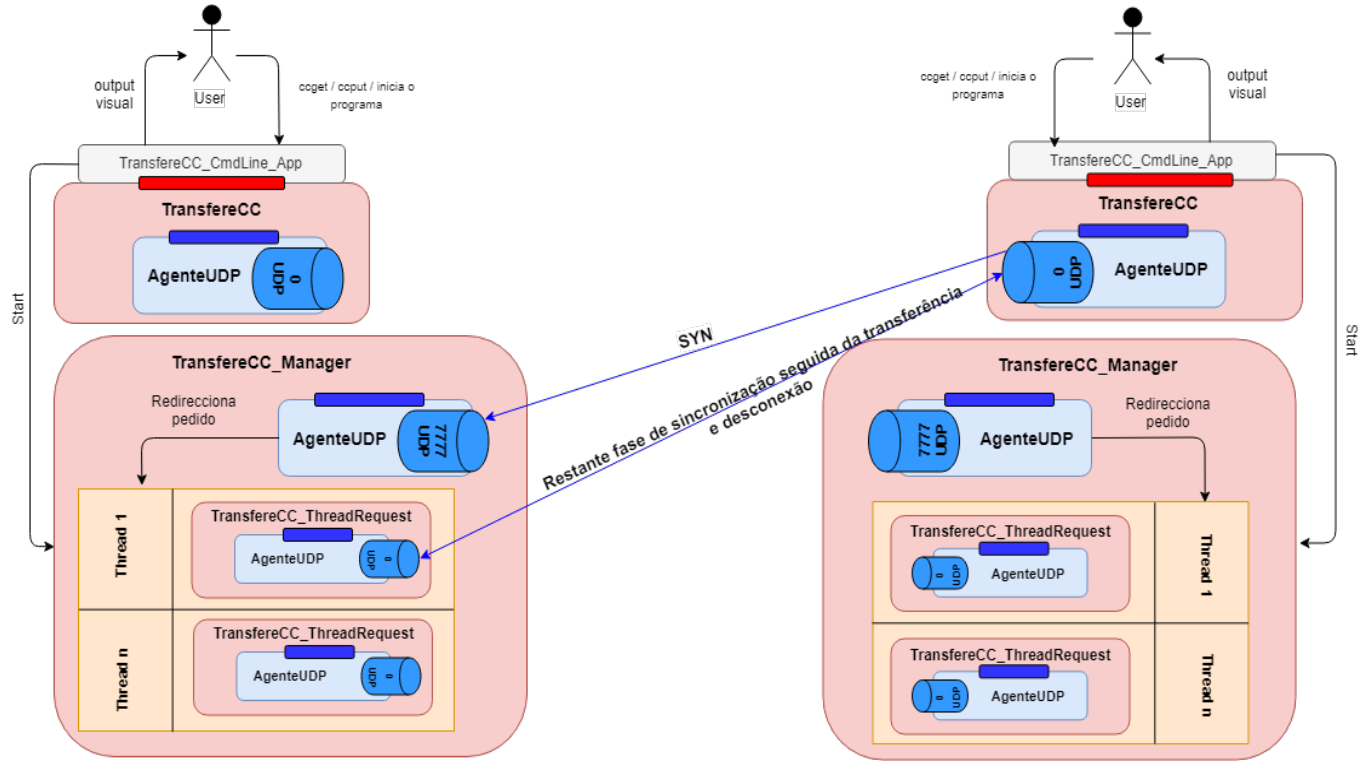

Figura 6. Esquema da arquitectura da solução.

O utilizador inicia a aplicação da linha de comandos *TransfereCC_CmdLine_App*, por sua vez, esta inicia o *TransfereCC* e o *TransfereCC_Manager*, que fica a fazer "*listening*"de pedidos externos.

O *TransfereCC* abre um *AgenteUDP* numa porta livre dada pelo sistema operativo (por definição porta 0 no java) e fica ao serviço do utilizador para este realizar pedidos.

O *TransfereCC_Manager* fica à espera de pedidos externos numa porta fixa (7777). E aquando da recepção de um pedido direciona esse pedido para o *TransfereCC_ThreadRequest* e é essa thread que trata do pedido.

De forma a utilizar a aplicação o user poderá executar os seguintes comandos.

```
TransfereCC > help
************************ HELP ************************
*                                                    *
*    SYNOPSIS:                                       *
*            ccget [OPTIONS] <filename> <IPDEST>     *
*            ccput <filename> <IPDEST>               *
*                                                    *
*    COMMANDS:                                       *
*            ccget - used to download a file         *
*            ccput - used to upload a file           *
*                                                    *
*    OPTIONS:                                        *
*            -name <localfilename>                   *
*                  This option is used to define the *
*                local file name if used on ccget    *
*                command.                            *
*                                                    *
*        In <filename> should be specified the name  *
*    of the file that you want to download or        *
*    upload.                                         *
*        In <localfilename> should be specified the  *
*    local name of the file to download.             *
*                                                    *
******************************************************
```

Figura 7. Comando help disponibilizado pela aplicação .

5 Testes e Resultados

De forma a testar a fiabilidade da implementação foram realizados vários testes à aplicação. Começou por se realizar testes em ambiente *localhost* uma transferência de ficheiros txt. Ao fim de vários resultados congruentes com o esperado passaram-se a realizar ensaios entre duas máquinas, em que uma delas se ligava à outra que servia de hotspot. Concluiu-se que estes resultados também estavam correctos. Assim sendo, escasso o tempo até à entrega deu-se o trabalho como terminado.

6 Conclusões

Em suma, o grupo de trabalho cumpriu maioria dos objectivos básicos com sucesso apesar de algumas dificuldades no que toca à decisão da arquitectura a adoptar. A compreensão de protocolos já existentes e a respectiva adaptação para a implementação também se revelou uma dificuldade para o grupo, assim como a sincronização de pacotes.

Contudo, o grupo reconhece que o trabalho possui falhas e que poderia ser melhorado e optimizado em vários aspectos, principalmente em deixar de funcionar segundo um algoritmo *Stop-and-Wait* e passar a implementar as funcionalidades adicionais.

Por fim, este trabalho revelou-se crucial para o entendimento e consolidação de matérias relativas à camada de transporte (4) e de aplicação (7) da pilha protocolar, entre outros.

Referências

1. MTU and MSS: What You Need to Know, https://www.imperva.com/blog/mtu-mss-explained
2. RIPE NCC, https://labs.ripe.net/Members/gih/fragmenting-ipv6
3. Transmission Control Protocol, https://en.wikipedia.org/wiki/Transmission_Control_Protocol
4. User Datagram Protocol, https://pt.wikipedia.org/wiki/User_Datagram_Protocol