Transferência fiável de dados sobre UDP

João Leal, João Vieira e Manuel Monteiro

Universidade do Minho

Abstract. O presente documento foi realizado no âmbito da disciplina de *Comunicações por Computador* do MIEI, relativo ao trabalho prático nº2, que aborda a implementação de um *Serviço de transferência rápida e viável de dados sobre UDP*. O objetivo principal para este trabalho passa por implementar um serviço de transferência de ficheiros sobre uma conexão **UDP**, definindo um protocolo de transferência fiável. Em primeiro lugar, foi criado o agente que irá receber/enviar os ficheiros de dados, e definiu-se o formato das **PDU**. Sendo assim, foi realizada a implementação do protocolo de transferência, com controlo de conexão, receção ordenada de tramas e controlo de erros. A informação para cada agente, das suas trocas de ficheiros, são guardadas numa tabela de estado. No final, são realizados testes para validar a solução construída, assim como as respetivas conclusões sobre os resultados.

1 Introdução

O primeiro passo na resolução deste trabalho prático, passou por decidir a linguagem de programação em que estaria implementado o serviço. O grupo optou por utilizar *python*, pois possui uma *syntax* simplificada, inúmeras bibliotecas que facilitam toda a conceção do serviço, mas tendo também o objetivo de aumentar o conhecimento sobre a linguagem.

De seguida, é necessário desenhar a arquitetura da aplicação, com os seus componentes em diferentes camadas lógicas, e com a criação de uma forma de comunicação viável. Após isto, surgem todos os desafios relativos à elaboração do protocolo de transferência, desde o formato das PDU, até às interações que irão existir para assegurar uma transferência do ficheiro segura.

Desta forma, ao longo deste relatório será exposto e explicado todo o processo que foi percorrido até ao resultado final.

2 Arquitetura da Solução

A arquitetura estabelecida tem por base a arquitetura inicialmente proposta no enunciado. Como tal, as diferentes camadas da solução são definidas da mesma forma, e a comunicação entre os agentes é feita utilizando *threads*, que operam sobre os agentes UDP:

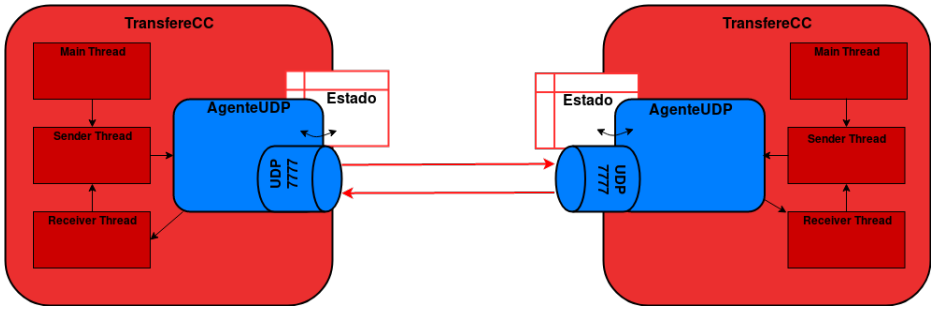


Fig. 1. Arquitetura da aplicação

3 Especificação do protocolo

3.1 PDU - formato das mensagens protocolares

De maneira a armazenar os diferentes tipos de informação inerentes à transferência de ficheiros, é necessário criar a estrutura dos pacotes UDP. Os pacotes têm todos a mesma estrutura no formato *JSON*, sendo que a principal diferença será o tipo de cada, dependendo da informação a enviar. Como tal, os diferentes campos da nossa **PDU** são os seguintes:

- **Type** - Indica de que tipo é o pacote;
- **Sequence** - Nº de sequência a que corresponde o pacote;
- **Offset** - Valor inteiro que indica quantos pacotes de um mesmo tipo faltam ser enviados;
- **Checksum** - Valor calculado para o pacote, necessário para verificar a integridade dos dados;
- **Data** - Campo que corresponde aos dados a enviar pelo pacote.

3.2 Interações

A comunicação entre os dois clientes, é feita em *unicast*, em que um deles efetua a conexão inicial e o outro apenas tem de aceitar ou recusar a mesma antes de transferir ficheiros entre os mesmos. Depois ambos podem adicionar ficheiros para o outro cliente transferi-los. Ao enviar cada pacote é calculado o seu *checksum* de modo a que quem o receba possa validar o pacote recebido, além de que é utilizado um mecanismo com *slinding window*, para a retransmissão de pacotes. Recebendo e validando o pacote, é enviado o respetivo *ACK* do pacote, de forma a que o cliente a enviar verifique que o pacote foi enviado, podendo assim mudar a janela de retransmissão dos pacotes. Para cada pacote representante de um ficheiro, é actualizada a tabela de estados, com o número de pacotes recebidos/perdidos.

4 Implementação

É importante referir que devido às dificuldades encontradas na implementação do serviço e da transferência, apenas foram implementadas as funcionalidades base obrigatórias.

4.1 PDU

Esta componente contém a estrutura dos pacotes UDP, como já foi referido, trata-se de uma entrada de dicionário (*JSON*) com os respetivos campos da PDU.

4.2 agenteUDP

A entidade que realiza toda a comunicação, enviando e recebendo pacotes UDP é a componente agenteUDP. Esta tem definida a estrutura do agente, com um *socket* UDP para receber conexões, e várias informações como o seu endereço IP e a sua porta assim como as do agente destinatário, e o seu estado de conexão. Também é nesta camada que é feito o envio e receção dos pacotes pelo *socket*, realizando a respetiva codificação e descodificação dos mesmos. A função *receivePacket* recebe pacotes com tamanho máximo 1500 bytes, de cada vez.

```
def sendPacket(self,packet):
    str = json.dumps(packet.packet)
    _,ready,_ = select.select([],[self.agentSock],[])
    if ready[0]:
        self.agentSock.sendto(str.encode("utf-8"), (self.send_addr, self.send_port))
    print ("Packet Sent: " + str + "\n")

def receivePacket(self):
    ready = select.select([self.agentSock],[],[])
    packet = 0
    address = ""
    if ready[0]:
        packet,address = self.agentSock.recvfrom(1500)
        packet = json.loads(packet.decode())
    return packet,address
```

Fig. 2. *Funções sendPacket* e *receivePacket*

4.3 TransfereCC

Este módulo implementa as funcionalidades necessárias ao bom funcionamento da transferência de forma a que esta ocorra como pretendido. Assim sendo, foi implementado um mecanismo ***checksum*** de validação de integridade do pacote recebido, de forma a garantir a fiabilidade da transferência e evitar a recepção de pacotes erróneos. Para o cálculo deste valor *checksum* começamos por verificar o tamanho em bytes da *data* a ser enviada e se este tomar um valor

ímpar, adicionamos um byte a zero (0x00) no fim, fazendo o mesmo para o IP de destino e o IP da fonte, aquando do cálculo deste valor. Em seguida, iteramos pelos bytes da data e dos dois IP retirando 2 bytes de cada vez e deslocando o primeiro byte, oito bits à esquerda, e adicionando o segundo byte, obtendo como consequência um long de 2 bytes. Estes valores serão somados em cada iteração, sendo que os casos de excesso são tratados de forma a que a checksum se mantenha um long de 2 bytes. No final, invertemos todos os bits da checksum e retiramos os últimos 16 bits como checksum final. Aquando da receção de um pacote, este valor é calculado novamente e comparado ao do pacote recebido.

Além do mecanismo checksum, implementamos também um mecanismo de confirmação ***ACK*** e retransmissão, no caso de um pacote perdido. Para tal, no início da execução do programa, são criadas duas threads auxiliares, além da *main* thread que apenas vai ouvir os pedidos que o utilizador invoca na linha de comandos. Já as auxiliares, uma vai receber os pacotes, atuando de forma diferente para cada tipo de pacote, já a outra apenas recebe numa fila de espera, quaisquer pacotes que as outras duas threads possam criar para ser enviados, encarregando-se assim apenas de envia-los. Tudo isto é efetuado no mesmo agente UDP, apenas as threads que estão a ouvir/enviar da mesma socket, só o fazem se houver pacotes a serem recebidos/enviado na fila, para não bloquearem o acesso à socket. Desta forma podemos enviar pacotes enquanto recebemos os respetivos ***ACK*** por exemplo.

Já para a retransmissão, foi criado um mecanismo de *Sliding Window* que utiliza um temporizador criado por nós. A thread que envia uma lista de pacotes, cria uma janela de quatro ou menos pacotes, dependendo do tamanho da lista, enviando os pacotes da janela e inicializando o temporizador. Se o temporizador chegar ao fim, os pacotes são reenviados. Porém o relógio pode ser parado pela thread que recebe pacotes, caso esta receba o respetivo ***ACK*** dos pacotes enviados. Assim os pacotes já não são reenviados, e a janela de envio é alterada, para mandar o resto dos pacotes da lista. Por fim, é criada uma última thread auxiliar, para enviar qualquer ficheiro a um cliente quando este último faz o pedido de transferência do mesmo, por forma a não parar a thread que recebe.

4.4 statusTable

A *statusTable* é uma lista de dicionários, que permite guardar informação sobre as transferências de ficheiros. Contém também todos os métodos necessários para controlo e atualização da tabela de estado. Os campos de entrada de cada dicionário são os seguintes:

```
dict = {"Type" : type,
        "File" : file_name,
        "IP Origem" : ip_orig,
        "Porta Origem" : port_orig,
        "Numero de pacotes" : n_packets,
        "Pacotes recebidos" : 0,
        "Pacotes perdidos": 0}
```

Fig. 3. Argumentos de entrada da tabela de estado

4.5 timer

Esta classe tem estruturado um temporizador, necessário à implementação da transferência na componente **TransfereCC**. O temporizador tem um tempo de início e duração, e são definidos os métodos necessários ao seu funcionamento tais como o *start* e *stop*.

```
def start(self):
    if self.start_time == self.TIMER_STOP:
        self.start_time = time.time()

def stop(self):
    if self.start_time != self.TIMER_STOP:
        self.start_time = self.TIMER_STOP
```

Fig. 4. Funções *start* e *stop*

5 Testes e Resultados (CORE)

Comandos:
put_file file - Faz *upload* do ficheiro "file"
get_file file - Faz *download* do ficheiro "file"
get ls - Recebe tabela de ficheiros do destinatário
ls - Recebe a própria tabela de ficheiros

5.1 Sem erros provocados

Ligação entre o nó Servidor1 e Cliente1:

Servidor1:
Aceita conexão do cliente1 e faz upload de um ficheiro "large.txt".

Final da transferência da parte do servidor1.

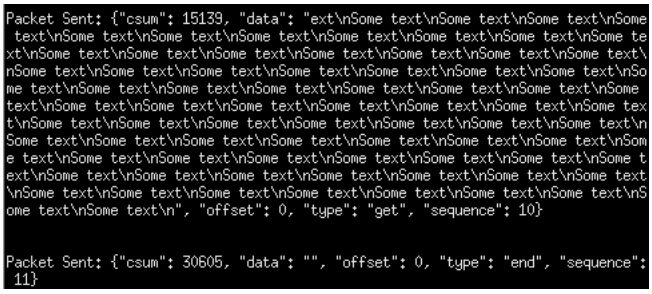
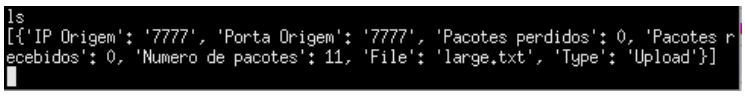


Tabela de estado do servidor1 após transferência:



Cliente1:
Faz a conexão com o servidor1 e pede download do ficheiro disponibilizado.

Final da transferência da parte do cliente1.

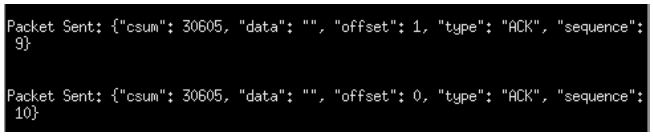
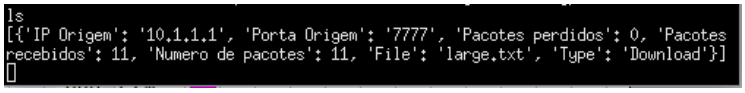


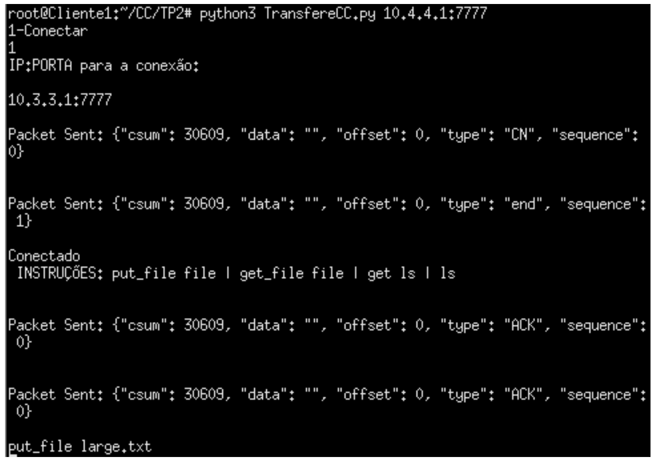
Tabela de estado do cliente1 após transferência:



5.2 Com erros provocados

Ligação entre o nó Cliente1 e Alfa, e transferência do mesmo ficheiro:

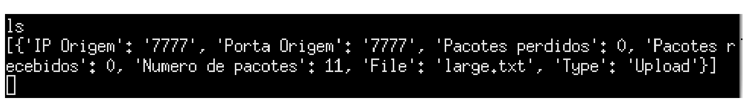
Cliente1:
Faz a conexão com o nó Alfa e faz upload do ficheiro "large.txt".



```
root@Cliente1:~/CC/TP2# python3 TransfereCC.py 10.4.4.1:7777
1-Conectar
1
IP:PORTA para a conexão:

10.3.3.1:7777

Packet Sent: {"csum": 30609, "data": "", "offset": 0, "type": "CN", "sequence":
0}


Packet Sent: {"csum": 30609, "data": "", "offset": 0, "type": "end", "sequence":
 1}

Conectado
 INSTRUÇÕES: put_file file | get_file file | get ls | ls


Packet Sent: {"csum": 30609, "data": "", "offset": 0, "type": "ACK", "sequence":
 0}


Packet Sent: {"csum": 30609, "data": "", "offset": 0, "type": "ACK", "sequence":
 0}

put_file large.txt
```

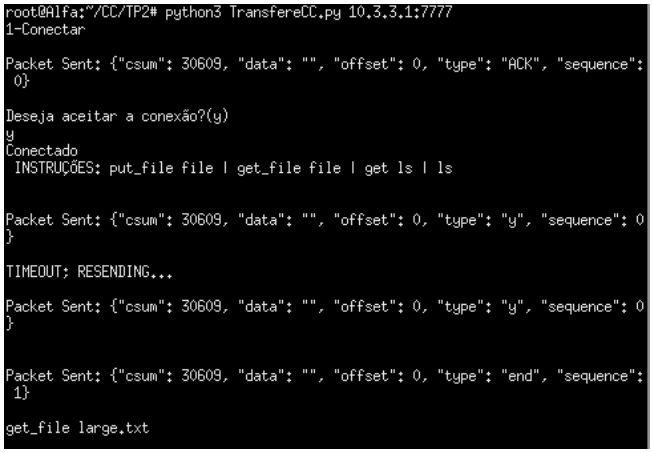
Final da transferência e tabela de estado.

```
text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome tex
t\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\n
Some text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSom
e text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome t
ext\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text
\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nS
ome text\nSome text\n", "offset": 0, "type": "get", "sequence": 10}

TIMEOUT; RESENDING...

Packet Sent: {"csum": 15143, "data": "ext\nSome text\nSome text\nSome text\nSome
 text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome te
xt\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\
nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSo
me text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome 
text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome tex
t\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\n
Some text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSom
e text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome t
ext\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text
\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nSome text\nS
ome text\nSome text\n", "offset": 0, "type": "get", "sequence": 10}


Packet Sent: {"csum": 30609, "data": "", "offset": 0, "type": "end", "sequence":
 11}
```

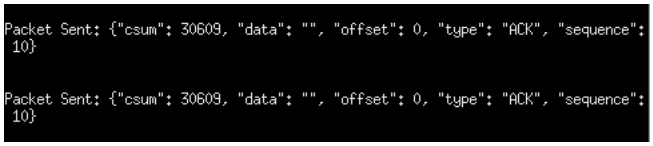


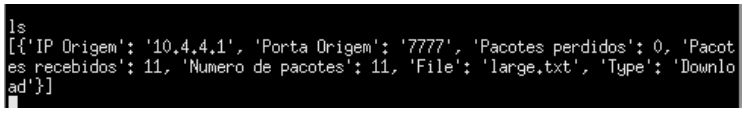
```
ls
[{'IP Origem': '7777', 'Porta Origem': '7777', 'Pacotes perdidos': 0, 'Pacotes r
ecebidos': 0, 'Numero de pacotes': 11, 'File': 'large.txt', 'Type': 'Upload'}]
```

Alfa:
Aceita conexão do cliente1 e faz download do ficheiro disponibilizado.



Final da transferência e tabela de estado.





Como é possível ver pelos *prints*, o serviço transfere um ficheiro com sucesso tanto numa ligação sem erros, assim como com erros.

Apenas denotar que, na ligação com erros provocados, são feitas algumas retransmissões dos pacotes mas o ficheiro é transferido na totalidade.

6 Conclusões

Com base nos resultados obtidos, o grupo cumpriu os principais objetivos deste trabalho, que se traduzia na implementação de um serviço de transferência de ficheiros sobre uma conexão UDP genérica, ficando também a perceber as funcionalidades necessárias que complementam uma transferência fiável. Sendo assim, apesar de alguma dificuldade na conceção da aplicação, foi possível a consolidação dos conceitos sobre transferência de dados através de serviços de transporte, abordados nas aulas, assim como a expansão das nossas bases de programação para uma linguagem fora da nossa zona de conforto (*Python*).