Relatório Trabalho Prático nº2

João Pimentel (a80874) Rodolfo Silva (a81716)
Pedro Gonçalves (a82313)

Braga, maio de 2019

Resumo

O protocolo de transferência UDP trata-se de um protocolo rápido, sem ordenação de pacotes e controlo de erros. Sendo o protocolo TCP o seu oposto, garantindo transferências fiáveis e seguras, apesar de mais lentas.

O objetivo deste projeto é o desenvolvimento de uma aplicação de transferência de dados baseada no protocolo UDP, mas com as características seguras do TCP, como controlo de conexão, controlo de erros, entre outros. Para tal, foram definidos *headers* para cada pacote de dados, possuindo números de sequência para garantir ordenação e um campo de *cheksum* para garantir integridade de cada pacote.

O projeto proporciona o acesso às características seguras e controladas do protocolo TCP, permitindo compreender os métodos de implementação das mesmas, a partir de um protocolo simples de envio/receção de dados.

Palavras-Chave: Header, TCP, UDP.

Universidade do Minho
Mestrado Integrado em Engenharia Informática
Comunicações por Computador
(2º Semestre/2018-2019)
Grupo 66

Conteúdo

Lista de Figuras

1 Introdução

O projeto teve como base a implementação de um clone do protocolo TCP a partir do protocolo UDP. Este último é um protocolo da camada de transporte, sendo caracterizado pela sua simplicidade e rapidez, não possuindo mecanismos de retransmissão, nem ordenação de pacotes.

De modo a conseguir obter um funcionamento semelhante ao protocolo TCP, foi necessário implementar mecanismos que garantissem a transmissão fiável de dados entre máquinas distintas através de conexões com perda e duplicação de pacotes, permitindo garantir que o documento enviado chegava num estado perfeito ao recetor.

A linguagem utilizada para a execução do projeto foi *Python*, graças à sua simplicidade e forma como permite implementar as estruturas de dados necessárias

Será, em seguida, explicada a forma como foi definido cada pacote de dados (PDU), a forma como foi implementado o controlo de conexão, consistência e ordenação na transferência dos ficheiros, destacando, ainda, os métodos de retransmissão de dados perdidos e validaçao de cada PDU.

2 Arquitetura da Solução

Para o correto desenvolvimento deste projeto, o grupo definiu que a arquitetura a implementar seguiria um certo modelo e regras, estando esta representada na Figura 1. É de notar que o grupo considera que a modularização efetuada permite uma fácil explicação de cada problema resolvido. Assim, tenham-se os seguintes módulos:

- **TransfereCC:** Este módulo representa o programa com o qual o utilizador interage para enviar ou receber um ficheiro;
- **Others:** Neste módulo está definido o *header* utilizado nas transferências e também a definição dos estados que gerem o servidor;
- **Servidor:** Possui toda a implementação relativa à receção de um ficheiro, e todos os processos necessários para o correto funcionamento da mesma;
- **Cliente:** Aqui encontra-se definindo o processo de envio de um ficheiro, incluindo todos os processos necessários para a garantia de um envio correto do mesmo;

- **Estado:** Este módulo retrata as tabelas informativas apresentadas no final de cada transfêrencia .

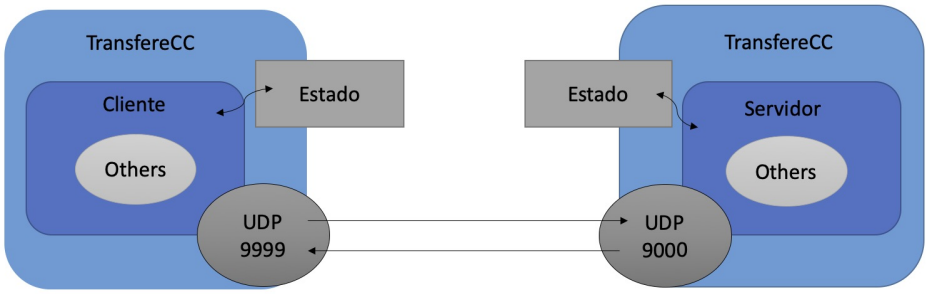


Figura 1 - Arquitetura do Projeto.

3 Especificação do Protocolo

3.1 Formato das mensagens protoculares (PDU)

Seguindo o protoclo TCP como modelo, o cabeçalho de cada PDU foi definido por uma sequência de campos como se mostra na Figura 2. Possui campos desde número de sequência até indicação de finalização de conexão. Dito isto, sejam explicados todos os campos:

- **Número de Sequência:** Número representativo da ordenação de cada PDU. Durante a transmissão de dados, o número presente neste campo indica ao sistema quais os *bytes* do ficheiro que este está a receber;
- **Número de Reconhecimento:** Divisão que garante que o pacote recebido é o esperado;
- **Checksum:** Campo utilizado para verificação da legitimidade dos dados recebidos. Para isso são usados os primeiros 16 *bits* do protocolo *SHA256* (algoritmo *Digest*) aplicado aos dados a transmitir. O mesmo é efetuado na chegada dos PDU, sendo o algoritmo aplicado aos dados, culminando numa comparação do valor obtido com o presente no PDU. Caso o valor seja diferente, o PDU é descartado;

- **Syn:** *Flag* que mostra que um dos lados da comunicação pretende estabelecer conexão com o outro;
- **Psh:** *Flag* que especifica que estão a ser transferidos dados entre os utilizadores;
- **Rst:** *Flag* que indica que o *pacote* está a ser reenviado;
- **Fin:** *Flag* que assinala a intenção de um dos lados da comunicação em terminar a conexão.

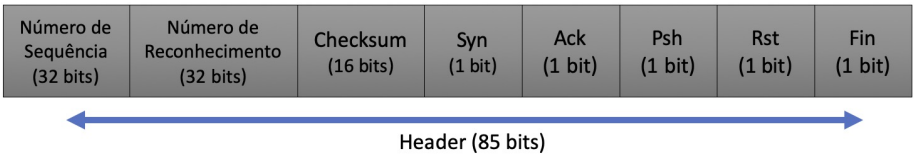


Figura 2 - Cabeçalho de um PDU.

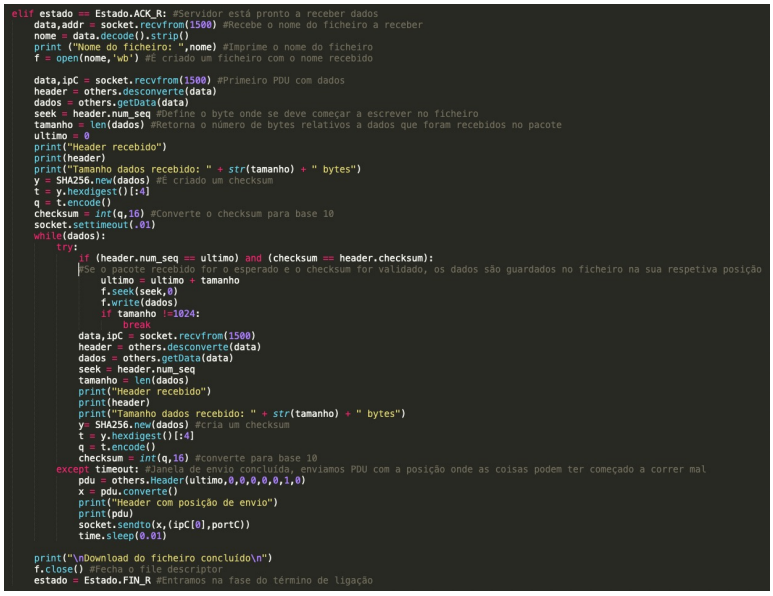
4 Implementação

Como mencionado anteriormente, o grupo optou pela linguagem de programação *Python*, mais especificamente a sua versão mais recente, pela simplicidade de implementação e eficiência em cenários cliente servidor.

4.1 Métodos e Algoritmos

As Figuras 3 e 4 representam os algoritmos referentes à receção e envio de dados, respetivamente. A implementação da transferência de um ficheiro foi baseada no método *Go Back N*, utilizando para isso uma janela com duração de 50 pacotes, ao fim dos quais é verificado a partir de que *byte* do ficheiro se vai voltar a enviar dados. Caso todos os pacotes sejam enviados, o envio será retomado no próximo pacote. No caso de algum pacote não chegar ao recetor, é começado o reenvio de todos os pacotes a partir do último pacote recebido com sucesso.

Note-se, ainda, que foram utilizados *timeouts*, de modo a permitir a sincronização de todo o processo, como se pode ver na Figura 3.

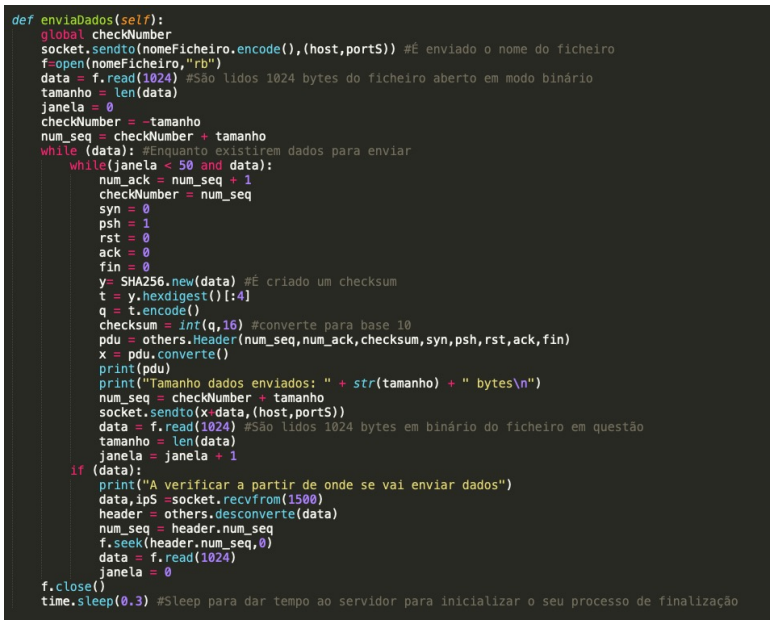


```
elif estado == Estado.ACK_R: #Servidor está pronto a receber dados
    data,addr = socket.recvfrom(1500) #Recebe o nome do ficheiro a receber
    nome = data.decode().strip()
    print ("Nome do ficheiro: ",nome) #Imprime o nome do ficheiro
    f = open(nome,'wb') #É criado um ficheiro com o nome recebido

    data,ipC = socket.recvfrom(1500) #Primeiro PDU com dados
    header = others.desconverte(data)
    dados = others.getData(data)
    seek = header.num_seq #Define o byte onde se deve começar a escrever no ficheiro
    tamanho = len(dados) #Retorna o número de bytes relativos a dados que foram recebidos no pacote
    ultimo = 0
    print("Header recebido")
    print(header)
    print("Tamanho dados recebido: " + str(tamanho) + " bytes")
    y = SHA256.new(dados) #É criado um checksum
    t = y.hexdigest()[:4]
    q = t.encode()
    checksum = int(q,16) #Converte o checksum para base 10
    socket.settimeout(.01)
    while(dados):
        try:
            if (header.num_seq == ultimo) and (checksum == header.checksum):
            #Se o pacote recebido for o esperado e o checksum for validado, os dados são guardados no ficheiro na sua respetiva posição
                ultimo = ultimo + tamanho
                f.seek(seek,0)
                f.write(dados)
                if tamanho !=1024:
                    break
            data,ipC = socket.recvfrom(1500)
            header = others.desconverte(data)
            dados = others.getData(data)
            seek = header.num_seq
            tamanho = len(dados)
            print("Header recebido")
            print(header)
            print("Tamanho dados recebido: " + str(tamanho) + " bytes")
            y= SHA256.new(dados) #cria um checksum
            t = y.hexdigest()[:4]
            q = t.encode()
            checksum = int(q,16) #converte para base 10
        except timeout: #Janela de envio concluída, enviamos PDU com a posição onde as coisas podem ter começado a correr mal
            pdu = others.Header(ultimo,0,0,0,0,0,1,0)
            x = pdu.converte()
            print("Header com posição de envio")
            print(pdu)
            socket.sendto(x,(ipC[0],portC))
            time.sleep(0.01)

    print("\nDownload do ficheiro concluído\n")
    f.close() #Fecha o file descriptor
    estado = Estado.FIN_R #Entramos na fase do término de ligação
```

Figura 3 - Algoritmo de Receção de um Ficheiro.



```
def enviaDados(self):
    global checkNumber
    socket.sendto(nomeFicheiro.encode(),(host,portS)) #É enviado o nome do ficheiro
    f=open(nomeFicheiro,"rb")
    data = f.read(1024) #São lidos 1024 bytes do ficheiro aberto em modo binário
    tamanho = len(data)
    janela = 0
    checkNumber = -tamanho
    num_seq = checkNumber + tamanho
    while (data): #Enquanto existirem dados para enviar
        while(janela < 50 and data):
            num_ack = num_seq + 1
            checkNumber = num_seq
            syn = 0
            psh = 1
            rst = 0
            ack = 0
            fin = 0
            y= SHA256.new(data) #É criado um checksum
            t = y.hexdigest()[:4]
            q = t.encode()
            checksum = int(q,16) #converte para base 10
            pdu = others.Header(num_seq,num_ack,checksum,syn,psh,rst,ack,fin)
            x = pdu.converte()
            print(pdu)
            print("Tamanho dados enviados: " + str(tamanho) + " bytes\n")
            num_seq = checkNumber + tamanho
            socket.sendto(x+data,(host,portS))
            data = f.read(1024) #São lidos 1024 bytes em binário do ficheiro em questão
            tamanho = len(data)
            janela = janela + 1
        if (data):
            print("A verificar a partir de onde se vai enviar dados")
            data,ipS =socket.recvfrom(1500)
            header = others.desconverte(data)
            num_seq = header.num_seq
            f.seek(header.num_seq,0)
            data = f.read(1024)
            janela = 0
    f.close()
    time.sleep(0.3) #Sleep para dar tempo ao servidor para inicializar o seu processo de finalização
```

Figura 4 - Método de Envio de um Ficheiro.

4.2 Bibliotecas de Suporte

Na implementação deste projeto foram utilizadas algumas bibliotecas merecedoras de referência, devido ao facto de estarem diretamente relacionadas com a área de desenvolvimento, transporte e proteção de dados.

- **Socket:** Biblioteca onde estão definidos todos os métodos utilizados para a criação e utilização de *sockets* UDP;
- **Crypto.hash:** Contêm os métodos utilizados pelo grupo para a definição do *checksum* através do protocolo *SHA256*.

5 Testes e Resultados

Após simples testes num ambiente de *localhost* (Figura 5), de modo a realmente testar o *software* desenvolvido, o grupo utilizou o emulador de redes CORE e a topologia fornecida pelos docentes da disciplina.

Foram enviados ficheiros que variavam desde simples ficheiros *.txt* até ficheiros *mp3*, de modo a comprovar a eficicácia na deteção e correção de possíveis erros nos pacotes transferidos, não tendo sido obtido nenhum ficheiro corrompido.

Em todos os testes realizados foram obtidos resultados satisfatórios, não só em termos de eficiência, mas também em termos de rapidez do processo, mesmo programando a rede para simular perdas de pacotes na ordem dos 50%.

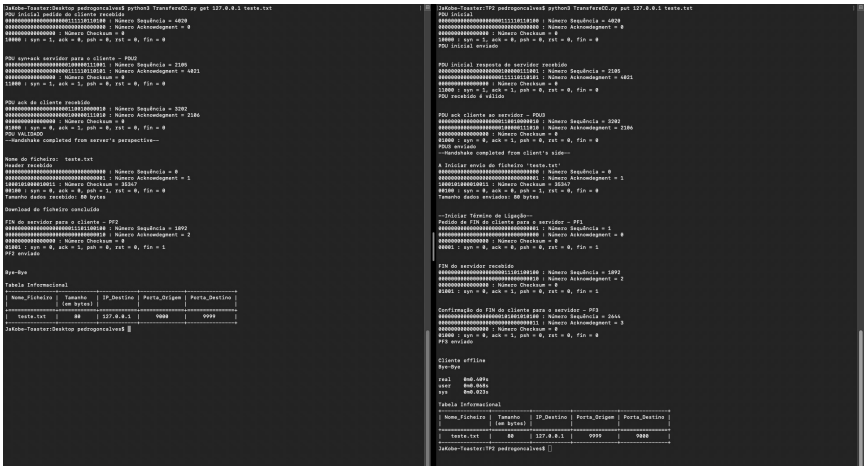


Figura 5 - Envio de Ficheiro em Ambiente *Localhost*.

6 Conclusões e Trabalho Futuro

O desenvolvimento deste projeto permitiu um aumento no conhecimento relativamente aos protocolos TCP e UDP.

Foi desenvolvida uma aplicação que fornece uma forma rápida e segura de transferência de dados permite que ficheiros de grandes dimensões sejam transferidos, através de redes com perda e/ou duplicação de pacotes, de forma segura, rápida e com a garantia de que chegarão ao destino intactos.

No entanto, existem aspetos a melhorar, nomeadamente a implementação de métodos de controlo de congestão, transferências multi-servidor, aumentar a segurança das transferências realizadas, entre outros aspetos.

Em suma, apesar de não ser perfeito, este projeto permitiu solidificar o conhecimento adquirido nas aulas teóricas, tendo sido bastante proveitoso para a possível realização de qualquer trabalho futuro relativo aos protocolos de transferência de dados.