UFCD 5118

Programação Orientada a Objetos - Introdução

Docente: João Galamba

Projecto - TFTPy

Formandos: Pedro Dores / Pedro Vieira

Indíce

Introdução e Objectivos

Este projeto foi a primeira oportunidade para os formandos aplicarem os conhecimentos adquiridos de Python num contexto de comunicação de redes, com o desenvolvimento de uma aplicação de transferência de ficheiros por protocolo TFTP entre máquinas virtuais em rede.

Para este propósito, foi feita a criação de programas de cliente e servidor TFTP, com as seguintes funcionalidades implementadas:

- Packing e unpacking de packets individuais
- Download e upload de ficheiros
- Cliente com modos de funcionamento interativo e não interativo
- Listagem de diretoria do servidor
- Seleção do porto pelo qual se faz a transferência
- Validação de hostname

Este trabalho requereu bastante investigação sobre o protocolo TFTP e aprendizagem sobre processos de comunicação em rede específicos para Python.

Também foi criado um repositório Git no endereço https://github.com/pv-tegrsi08/TFTPy para o propósito de partilha de código e gestão de versões dele.

Análise

O problema que se abordou neste projeto foi como coordenar dois computadores para o propósito de envio de ficheiros entre eles, tomando em consideração que eles não recebem atualizações em tempo real sobre o estado do processo de transferência de ficheiros entre eles e que não é possível transferir o ficheiro todo de uma vez. Isto requer o uso de um processo de divisão do ficheiro em blocos como e envio de mensagens de reconhecimento da transferência desses blocos.

Os fluxogramas seguintes apresentam uma ideia simplificada e de alto nível dos processos implementados neste projeto. Ambos apresentam uma estrutura básica com alguns componentes simples:

- Pedido de leitura/escrita de um ficheiro
- Envio de reconhecimento (quer seja de um pedaço de um ficheiro, ou de um pedido de escrita de um ficheiro)
- Envio de pedaço de ficheiro

Com a repetição alternada dos dois últimos pontos até à chegada do final do ficheiro.

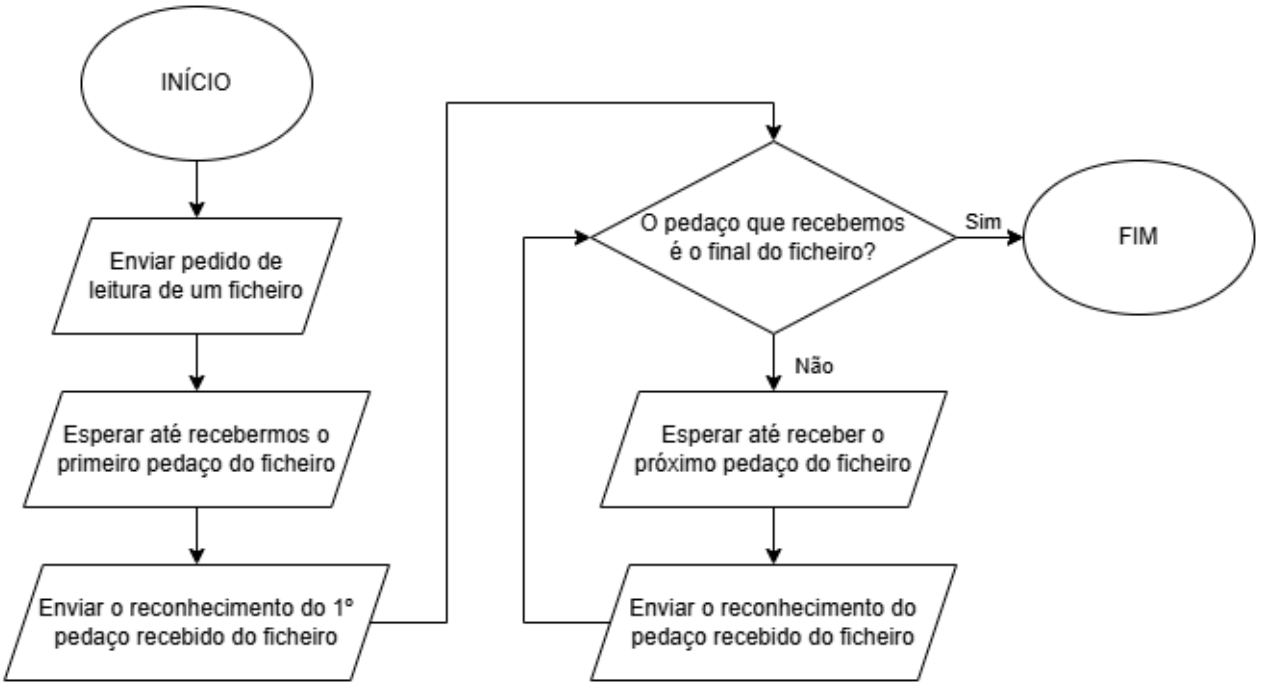

Imagem 1 - Fluxograma de alto nível - Processo do cliente para leitura de ficheiro

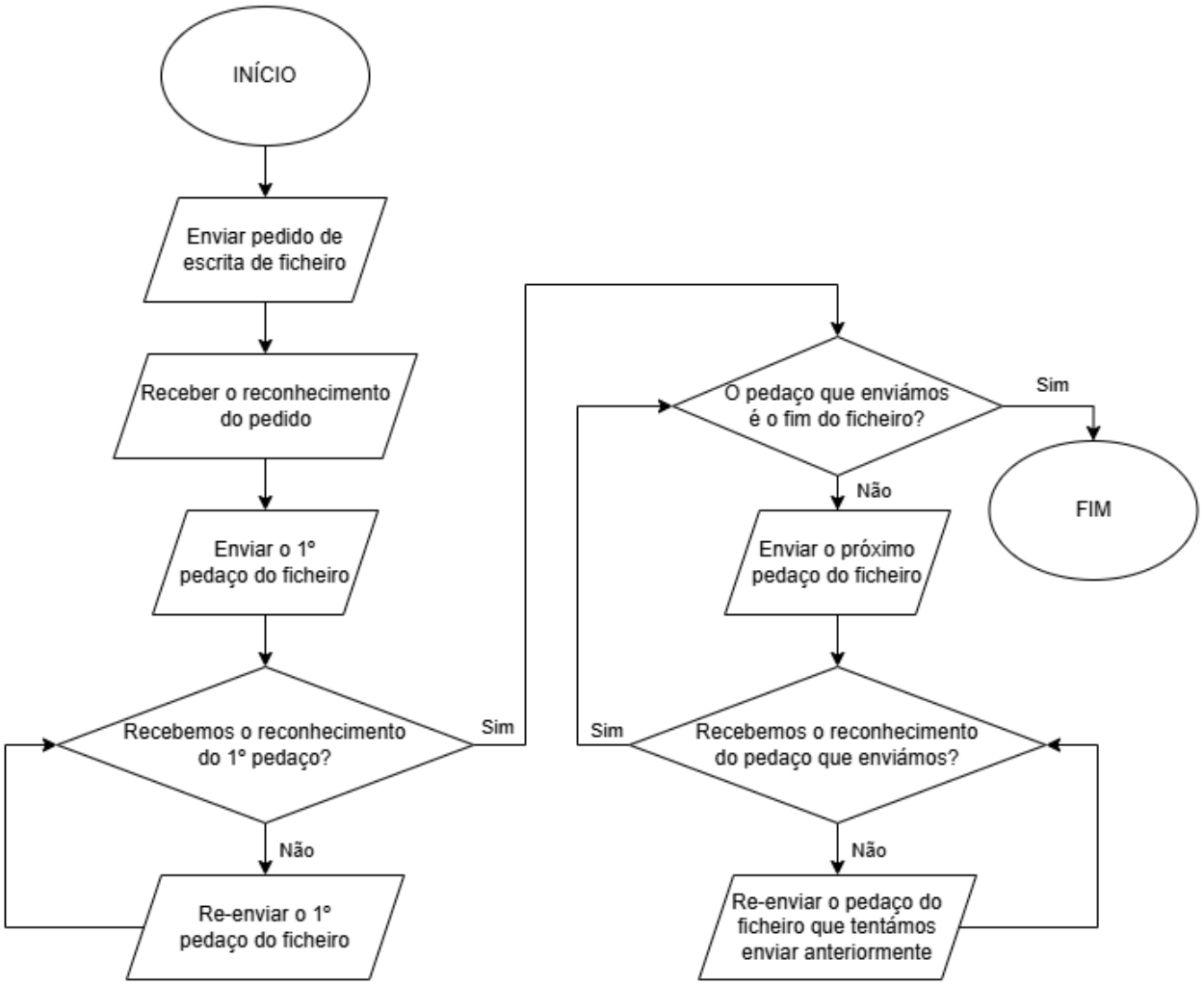

Imagem 2 - Fluxograma de alto nível - Processo do cliente para envio de ficheiro

Protocolo TFTP

O protocolo TFTP é um protocolo simples de transferência de ficheiros, publicado originalmente na IEN (Internet Experiment Note) nº 133 [1] em Janeiro de 1980, mas a publicação oficial currente do protocolo é a do RFC (Request For Comments) 1350 [2], publicada em Julho de 1992, a segunda revisão do protocolo. Ela não têm muitas das funcionalidades do protocolo FTP definidas no RFC 959 [3], como autenticação de utilizadores, criação e remoção de diretórios, controlo de formato vertical, etc…

Ele segue um sistema muito simples de mensagens numeradas, que são usadas no envio e receção de ficheiros em segmentos e reconhecimento dos segmentos reconhecidos. Os tipos de mensagens mais utilizados no protocolo são de pedido de leitura/escrita de ficheiro (estas não são numeradas), reconhecimento (*acknowledgement*) e envio de dados.

Os diagramas relacionados com a visualização dos processos de envio e recebimento de ficheiros serão apresentados nas próximas páginas.

[1] RFC Editor - “IEN 133 - The TFTP Protocol” Acessed on July 11, 2025. https://www.rfc-editor.org/ien/ien133.txt.
[2] IETF Datatracker. “RFC 1350: The TFTP Protocol (Revision 2).” Accessed on July 11, 2025. https://datatracker.ietf.org/doc/rfc1350.
[3] IETF Datatracker. “RFC 959: File Transfer Protocol,” Accessed on July 11, 2025. https://datatracker.ietf.org/doc/rfc959/.

Diagramas

Os seguintes diagramas de mensagens apresentam de modo mais detalhado o processo de leitura e escrita de ficheiros, com tamanho específico, de modo a poder-se visualizar o processo de numeração de pacotes *acknowledge* e de dados.

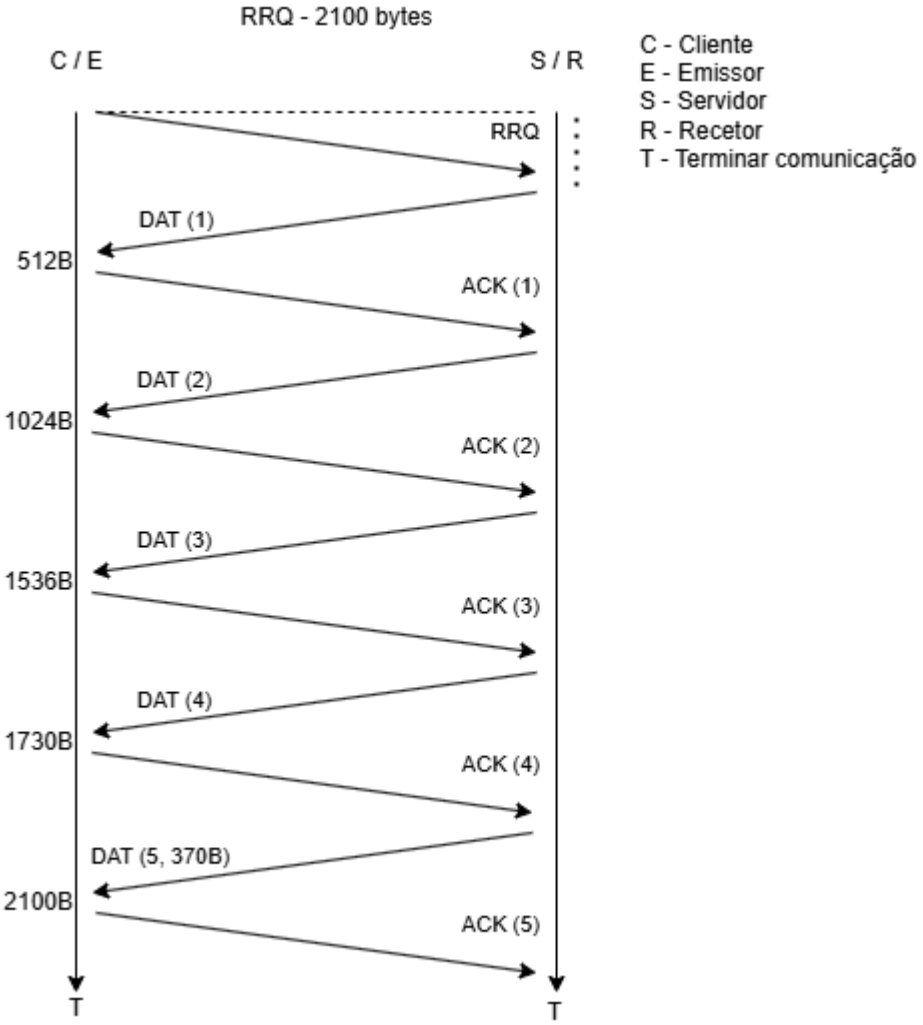

Imagem 3 - Diagrama de mensagens - Leitura de um ficheiro com 2100 bytes

Este diagrama mostra o processo de leitura de um ficheiro com um tamanho de bytes que não é divisível por 512 (o nº de bytes máximo para envio no protocolo TFTP), em que o último pacote DAT tem uma dimensão de 370 bytes. Após a receção do último pacote de dados (definido como o 1º pacote de dados com menos do que 512 bytes de dados nele), envia-se o *acknowledgement* final, que marca o fim deste processo.

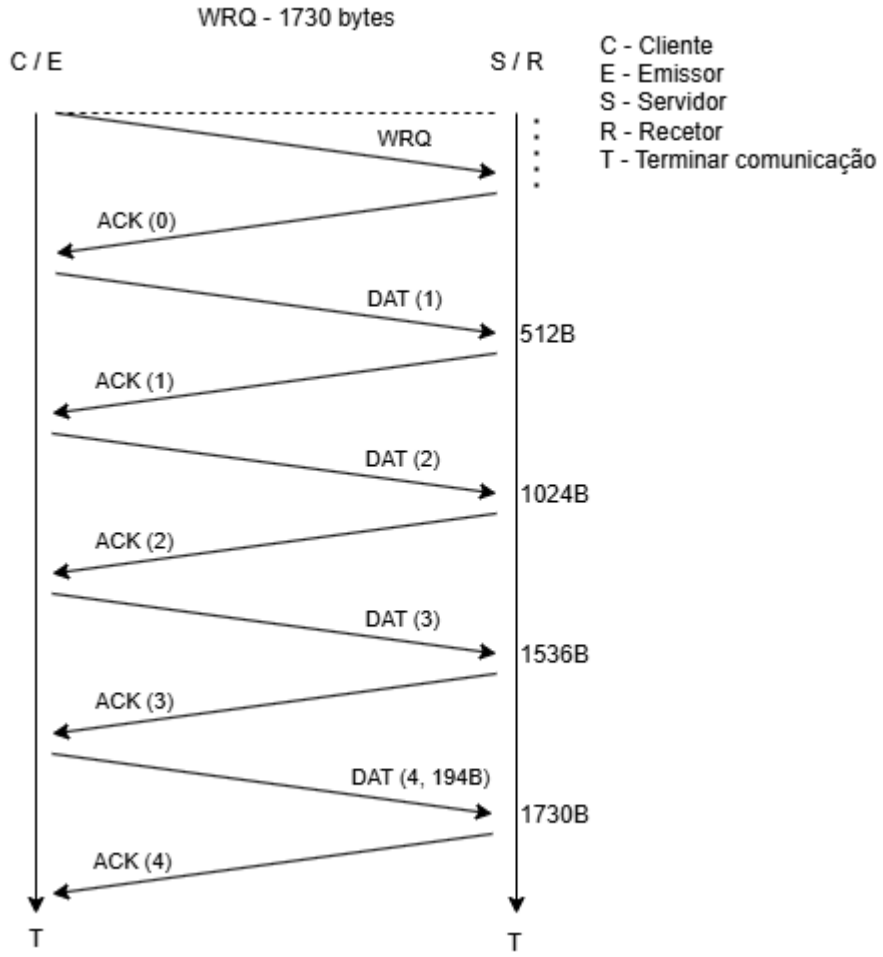

Imagem 4 - Diagrama de mensagens - Escrita de um ficheiro com 1730 bytes

Este diagrama mostra o processo de envio de um ficheiro com um tamanho de bytes que não é divisível por 512, em que o último pacote DAT enviado tem uma dimensão de 194 bytes. O servidor recebe o pacote, reconheçe que ele tem menos do que 512 bytes de dados nele e envia o último *acknowledgement*, marcando o fim deste processo.

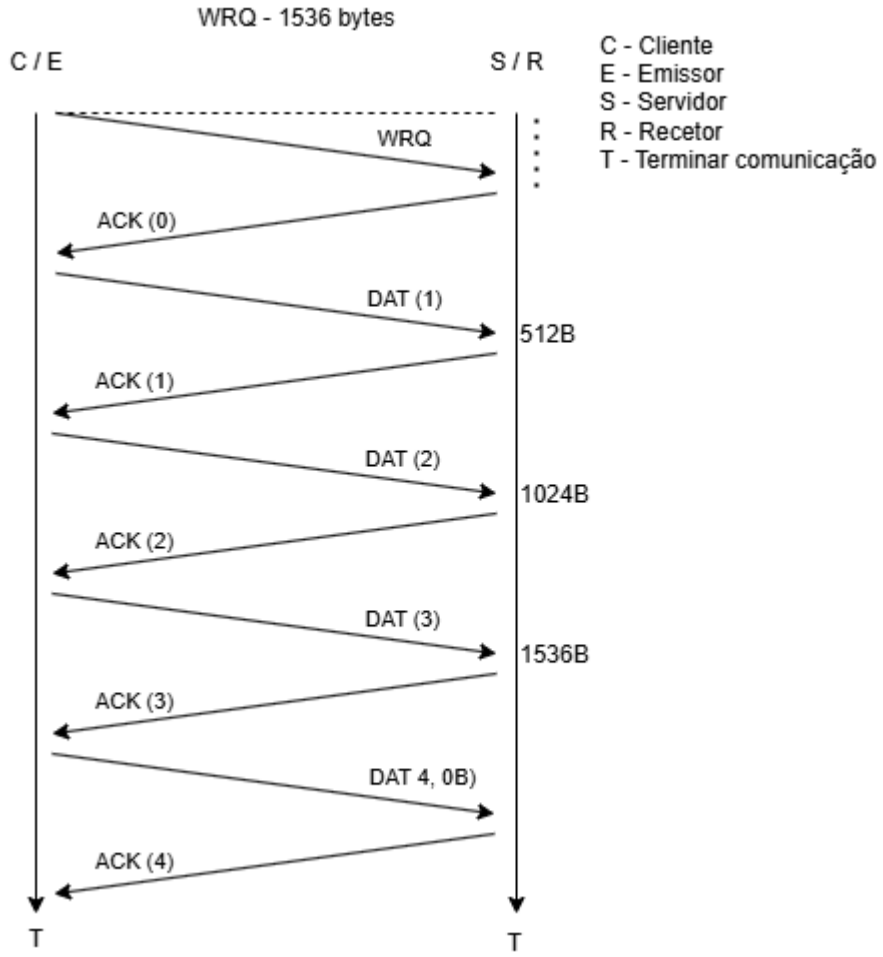

Imagem 5 - Diagrama de mensagens - Escrita de um ficheiro com 1536 bytes

Este diagrama mostra o processo de envio de um ficheiro com um tamanho de bytes que é divisível por 512 (1536/512 = 3), em que o último pacote DAT enviado tem uma dimensão de 0 bytes, o limite mínimo suportado pelo protocolo TFTP. O final do processo é idêntico ao apresentado na página anterior.

Formato de Pacotes

A formatação e ordenação de *headers* em pacotes é explicitada no RFC 1350[2], e esta deve ser seguida à risca, sem quaisquer desvios. Os diagramas abaixo apresentam os 4 principais tipos de pacotes processados no protocolo TFTP, com dimensões em bytes e ordenação de *headers* como indicada no RFC referida acima.

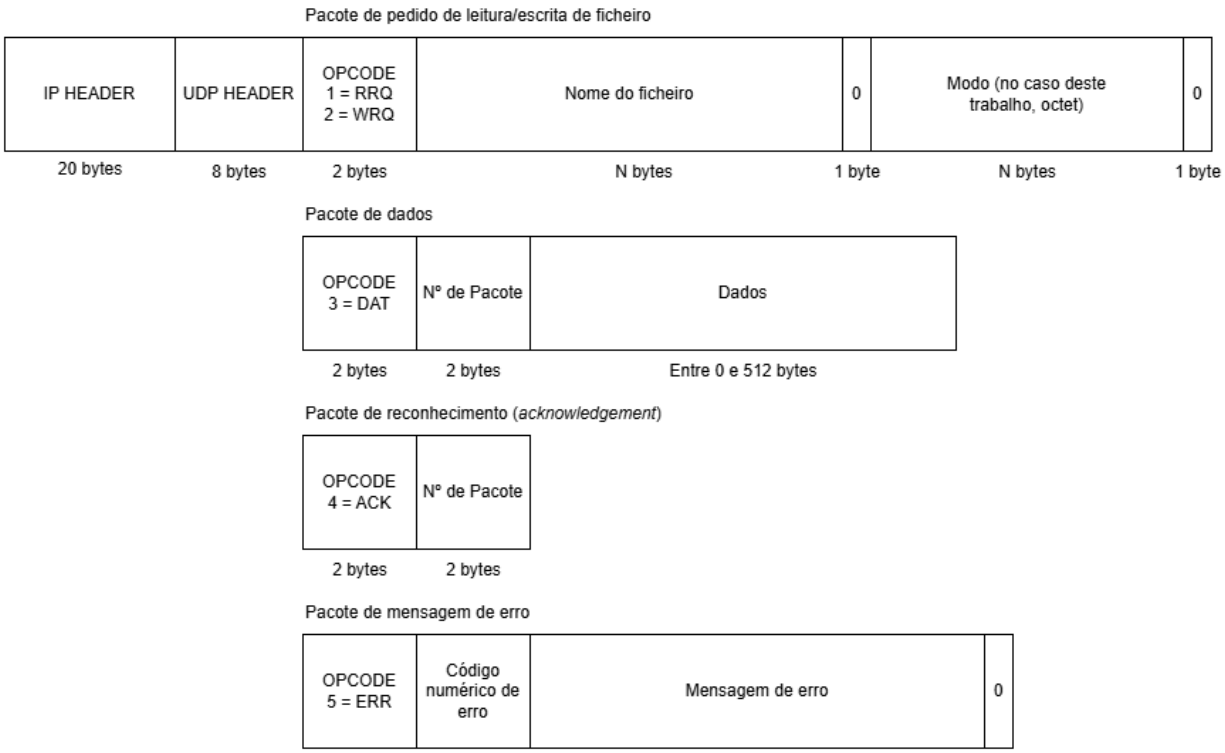

Imagem 6 - Principais tipos de pacotes reconhecidos no protocolo TFTP

Existem várias variáveis nestes pacotes, que só funcionam no protocolo TFTP com valores específicos:

- OPCODE
  - Pode ter valores entre 1 e 6 (Um valor de 6 representa um pacote OACK, não apresentado no diagrama acima, como parte do RFC 1782 [4], uma extensão do protocolo TFTP criada em Março de 1995)
- Nº de bloco
  - Entre 0 e 65535 (se o ficheiro precisar de mais do que 65535 pacotes de dados, o próximo pacote reinicia a contagem numérica a 1, já que o bloco nº 0 só é utilizado no primeiro *acknowlegment* de um pedido de escrita de ficheiro)

[4] IETF Datatracker. "RFC 1782: TFTP Option Extension," Accessed on July 11, 2025. https://datatracker.ietf.org/doc/rfc1782/.

- Código numérico de erro:
  - Entre 0 e 7
    - 0 - Não definido
    - 1 - Ficheiro não encontrado
    - 2 - Violação de acesso
    - 3 - Armazenamento cheio ou alocação de espaço excedido
    - 4 - Operação ilegal TFTP
    - 5 - ID de transferência desconhecido
    - 6 - Ficheiro já existente
    - 7 - Utilizador não existente
- Modo pode ter 3 valores:
  - 'octet', o único implementado neste trabalho
  - 'netascii', um modo só para texto
  - e 'mail', para transferência de emails

Estas variáveis são fundamentáveis no protocolo, e não se pode ignorar a importância delas no uso dele.

Desenho e Estrutura

Desenhou-se o programa tendo como objetivos principais a implementação da transferência de ficheiros, utilizando os conhecimentos sobre o protocolo TFTP adquiridos para este trabalho e os dois modos de funcionamento do cliente (não interativo e interativo). Mas planos para a implementação do servidor já foram logo formulados nas discussões entre os formandos deste trabalho. O servidor requer funcionalidades e processos próprios separados do cliente, então eles foram planeados da seguinte forma:

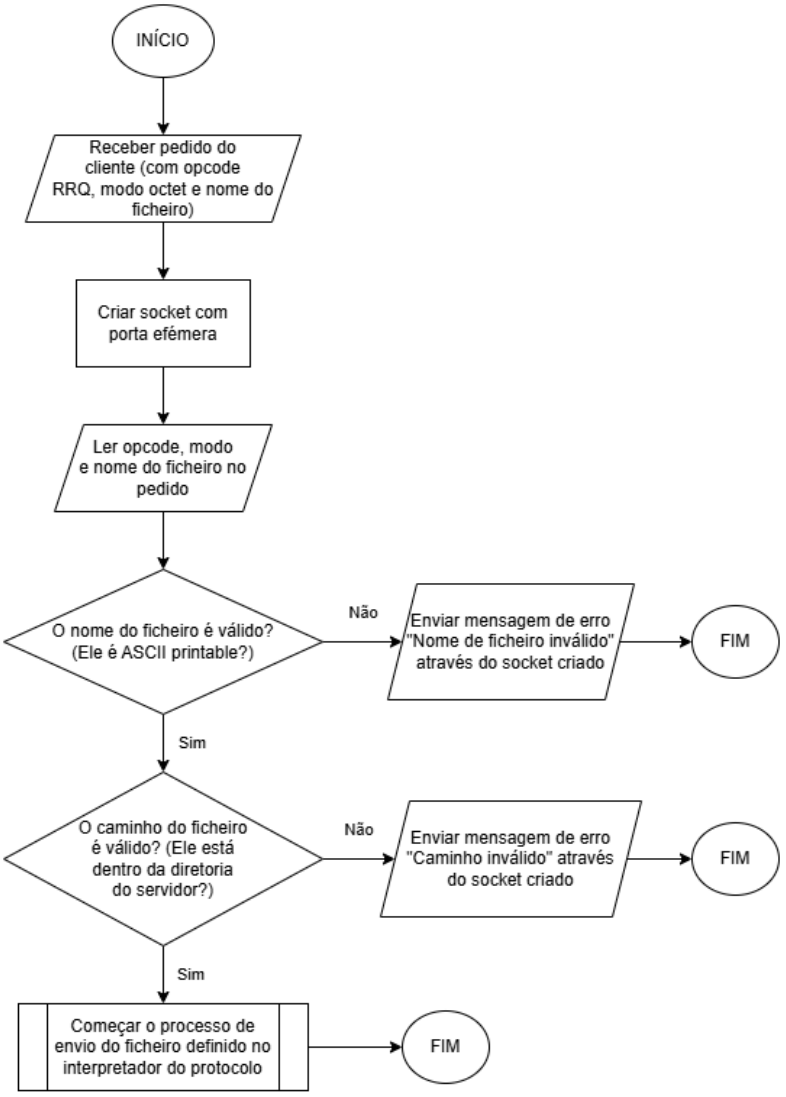

Imagem 7 - Fluxograma L1 - Processo de resposta do servidor a pedido de leitura de ficheiro (RRQ)

Estrutura geral

Dividiu-se o programa em três partes, sendo cada uma delas um ficheiro no repositório *git* criado para este trabalho:

- Cliente
  - Processamento de argumentos para ambos os modos de funcionamento (interativo e não interativo)
  - Criação e gestão de funcionalidades da shell interativa
- Servidor
  - Validação de pedidos
  - Criação de sockets para transferência de ficheiros
  - Listagem de diretoria
- Interpretador de protocolo
  - Definição de constantes do protocolo de acordo com as necessidades dele e as especificações do enunciado
  - Gestão das funções principais de transferência de ficheiros para ambos o cliente e o servidor
  - Empacotamento e desenpacotamento de pacotes
  - Gestão de erros nos pacotes e processos em execução

Originalmente havia uma quarta parte (um ficheiro com funções utilitárias para validação de endereços IP/*hostname*, verificar se os caracteres introduzidos na consola são *ASCII printable* e gerir a limpeza do ecrã no caso de uso de comandos *clear* ou *cls*) que acabou por ser combinada com o interpretador de protocolo, já que o enunciado indica que só devem existir os três ficheiros apresentadas acima no repositório final.

Implementação

A solução desenvolvida implementa um *subset* do protocolo TFTP em Python e é composta por três módulos principais: *client.py* e *server.py*, onde se encontram os interfaces do cliente e do servidor, e o módulo *tftp.py* com a lógica do protocolo.

Os pacotes TFTP (RRQ, WRQ, DAT, ACK, ERR) são gerados utilizando a biblioteca *struct*, garantindo o formato correto conforme o RFC 1350[2].

Os valores dos Opcodes foram guardados numa Enum, sugestão feita pelo Professor Galamba no vídeo de instrução, que nós aproveitámos.

O cliente foi implementado com a biblioteca *Cmd*, permitindo criar um shell interativo com os comandos *get, put, dir* e *quit*. Para melhorar a experiência do utilizador, foi adicionada a funcionalidade *clearScreen*, funcionando em multiplataforma. O utilizador pode assim interagir com o servidor de forma simples e intuitiva. Este código partilha funções com o modo não interativo (argumentos por linha de comando). Esta é uma funcionalidade opcional que foi implementada.

Os argumentos de linha de comando são obtidos com recurso a uma biblioteca externa, que requer instalação via *pip install docopt*.

A funcionalidade extra *dir* foi também implementada, podendo o cliente solicitar uma listagem do diretório remoto, gerada pelo servidor de forma multiplataforma (usando *dir* no Windows e *ls -lh* em Linux/macOS), guardada num ficheiro temporário e enviada reutilizando a lógica de transferência de ficheiros. No caso de o servidor não disponibilizar esta funcionalidade, enviará um erro que o cliente traduzirá para `'dir' is not supported by this TFTP server`.

O Servidor *multi-threaded* foi igualmente implementado, suportando vários pedidos em simultâneo, com cada pedido a ser tratado numa *thread* separada. Aqui usamos o módulo *threading* e o módulo *socket*, sendo a este último que recorremos para obter um porto efémero, libertando assim o servidor para receber e processar novos pedidos:
`transfer_sock.bind(('', 0))`

Foi também implementado um temporizador de retransmissão para pacotes *DAT*, tanto no servidor (envio de ficheiros e listagem) como no cliente (envio de ficheiros via *put*). Após o envio de cada bloco, o emissor espera pelo *ACK* durante um tempo limite (*INACTIVITY_TIMEOUT*). Se não receber o *ACK*, reenvia o mesmo bloco até um máximo de tentativas (*MAX_RETRIES*). Este mecanismo garante fiabilidade sobre *UDP*, conforme o enunciado.

Para transferências que ultrapassem os 65535 blocos (ficheiros com ~32 MB), o contador de blocos voltará a 1, permitindo trabalhar ficheiros maiores:
`block_number = (block_number) % 65535 + 1`

Adicionalmente, o servidor não permite solicitações de *get* e *put* a ficheiros fora do diretório principal, nem mesmo a subdiretórios. O cliente poderá enviar ficheiros de outras pastas mas para que sejam aceites pelo servidor, terá de fornecer o nome remoto sem subdiretórios:

```
$ python src/client.py put -p 12345 localhost subdir/ficheiro.pdf
TFTP error: Invalid filename.
$ python src/client.py put -p 12345 localhost subdir/ficheiro.pdf
Ficheiro.pdf
Sent file 'subdir/ficheiro.pdf' 122880 bytes.
```

E no servidor:

```
[16:00:00] Received file 'Ficheiro.pdf' from ('127.0.0.1', 42235)
```

O número de bytes transferidos é calculado com '*os.path.getsize()*' após a transferência. Este procedimento estava inicialmente a ser feito multiplicando o tamanho dos blocos (512b) pelo número de blocos mais os bytes do último bloco, mas apresentava imprecisões.

Pode correr o projeto em Windows da seguinte forma (os comandos em itálico devem ser executados numa *command prompt* normal, não no *powershell*):

1. Ter o Python instalado
2. Instalar o venv
   `python -m venv venv`
3. Ativar o ambiente virtual
   `venv\Scripts\activate`
4. Instalar o docopt
   `pip install docopt`
5. Correr o servidor
   `python src\server.py pasta 12345`
6. Para sair fazer *`Ctrl+C`*
7. Correr o cliente:
   `python client.py -p 12345 localhost`
8. Para sair escrever *`quit`*

Conclusão

O projeto implementa uma solução de transferência de ficheiros via TFTP e Python, suportando operações de leitura, escrita e listagem remota, suportando modo interativo e não interativo do cliente e múltiplos processos de clientes em concorrência no servidor.

O temporizador de retransmissão foi igualmente implementado para garantir a fiabilidade da transferência via UDP.

Este projeto permitiu aprender e explorar aspetos fundamentais em comunicação em rede, que muitas vezes nos passam despercebidos, em programação concorrente e em manipulação de ficheiros, conhecimentos essenciais para um profissional de Tecnologias de Informação.

Anexo I - UDP

A sigla UDP (User Datagram Protocol) refere-se a outro protocolo de transmissão de dados na camada de transporte do modelo OSI. A seguinte tabela apresenta-a em comparação com o TCP (Transmission Control Protocol):

	UDP	TCP
Pedido de sincronização e *acknowledgement* (SYN ->SYN-ACK -> ACK) no estabelecimento inicial da ligação entre 2 computadores:	Não	Sim
Uso de blocos numerados para ordenação de dados e *acknowledgement*	Não	Sim
Comparação de checksum dos dados entre o emissor e o recetor	Não	Sim
Re-envio de dados no caso de falta de *acknowledgement* para o bloco respetivo	Não (*Acknowledgement* não é utilizado neste protocolo)	Sim
Dimensão do *header* num bloco de dados	Menor	Maior
Utilizado para:	Streaming de dados em tempo real (VoIP, *gaming*, video de eventos ao vivo, etc…)	Transferência de documentos, páginas web, tudo o que requer integridade dos dados enviados e recebidos
Desenhado com que propriedades em mente:	Menor latência e uso de menos recursos computacionais	Fiabilidade e garantia de transmissão total de dados

Tabela 1 - Comparação de protocolos UDP e TCP

Como se pode ver, o protocolo UDP têm as suas vantagens e desvantagens, e a escolha de que protocolo utilizar (havendo muito mais protocolos do que estes dois) depende da situação que se têm em frente.

Webgrafia

[1] - RFC Editor - “IEN 133 - The TFTP Protocol” Acessed on July 11, 2025. https://www.rfc-editor.org/ien/ien133.txt.

[2] - IETF Datatracker. “RFC 1350: The TFTP Protocol (Revision 2),” Accessed on July 11, 2025. https://datatracker.ietf.org/doc/rfc1350.

[3] - IETF Datatracker. “RFC 959: File Transfer Protocol,” Accessed on July 11, 2025. https://datatracker.ietf.org/doc/rfc959/.

[4] - IETF Datatracker. “RFC 1782: TFTP Option Extension,” Accessed on July 11, 2025. https://datatracker.ietf.org/doc/rfc1782/.