TEGRSI12
Rodrigo Nunes
Henrique Saraiva
5119

PROJECTO 3 - TFTPy: CLIENTE TFTP

Tabela de Conteúdos

Introdução e Objectivos

O objectivo deste projecto é desenvolver uma aplicação de TFTP (Trivial File Transfer Protocol) para transferência de ficheiros e de assim modo aprender Python 3 para programar em redes utilizando sockets.

O protocolo TFTP é um protocolo simples de implementar e leve em memória para que um cliente (client) faça download ou upload de ficheiros para um servidor (host) usando o protocolo UDP (User Datagram Protocol).

A implementação do programa consistirá de vários scripts (client, funções partilhadas pelo client e o host, modulos externos como o docopt, etc.) utilizando sockets e Classes (Object Oriented Programming) entre outros.

Análise

Diagrama de envio de 1730 bytes

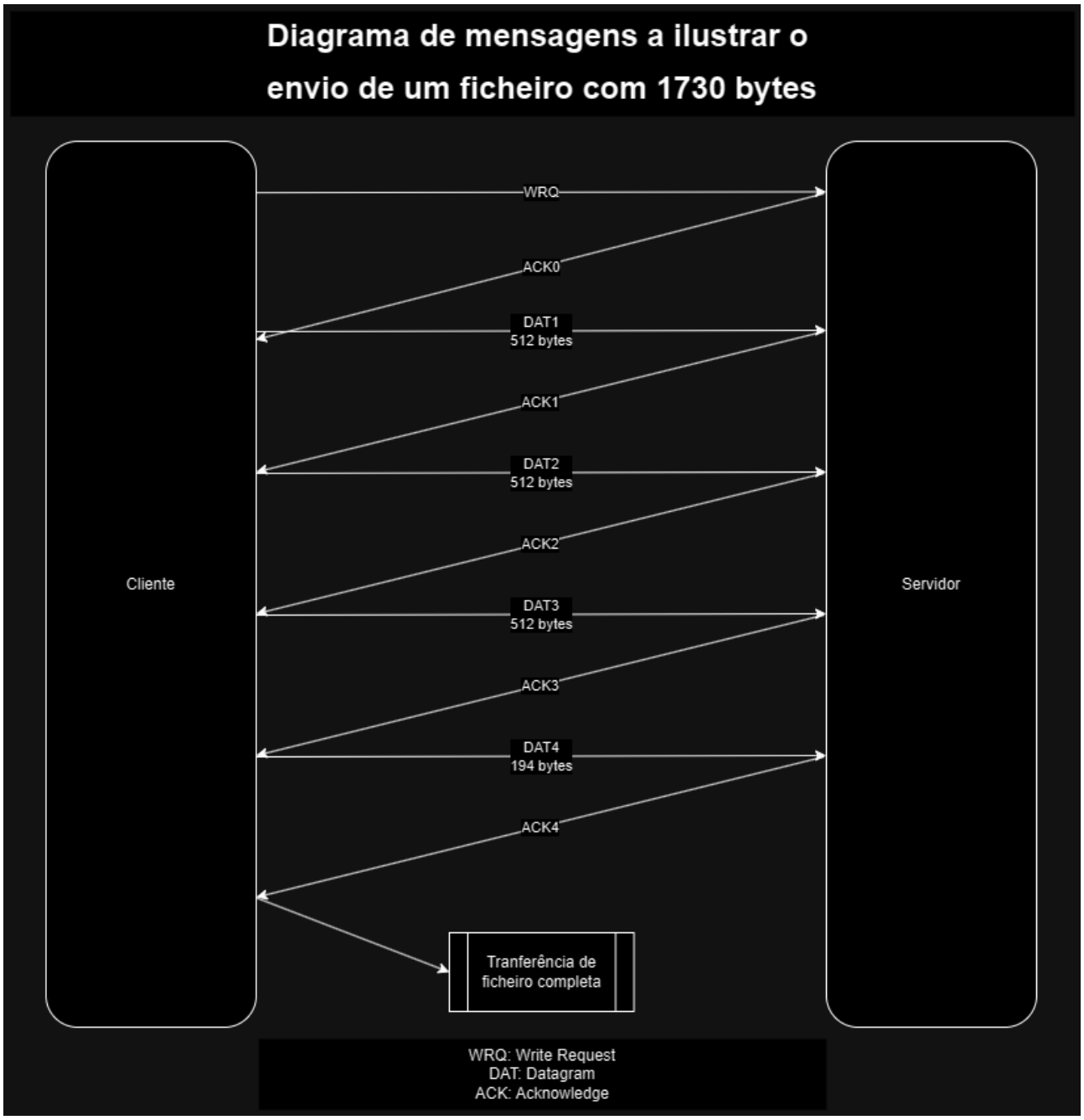


Diagrama de envio de 1536 bytes

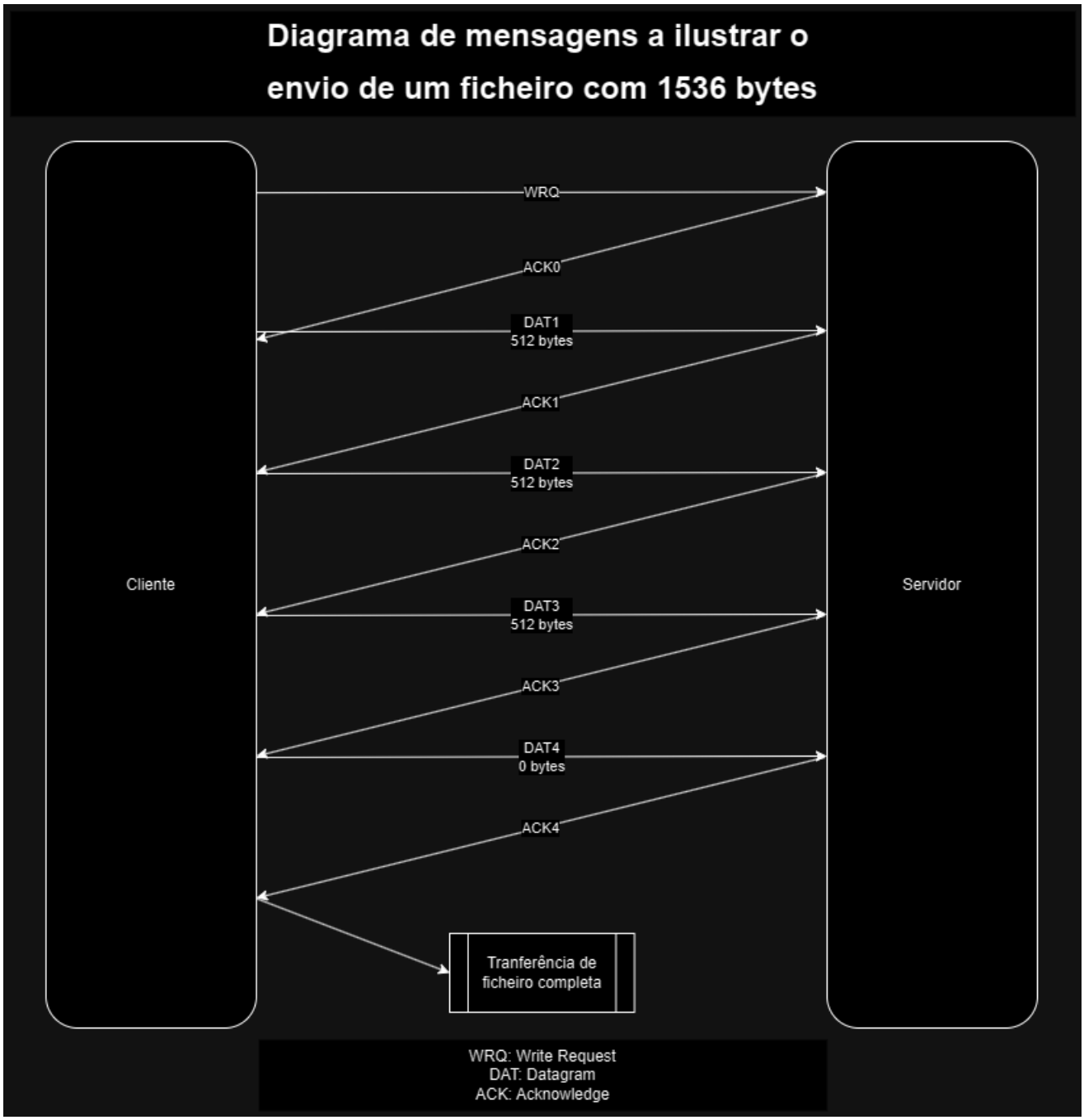
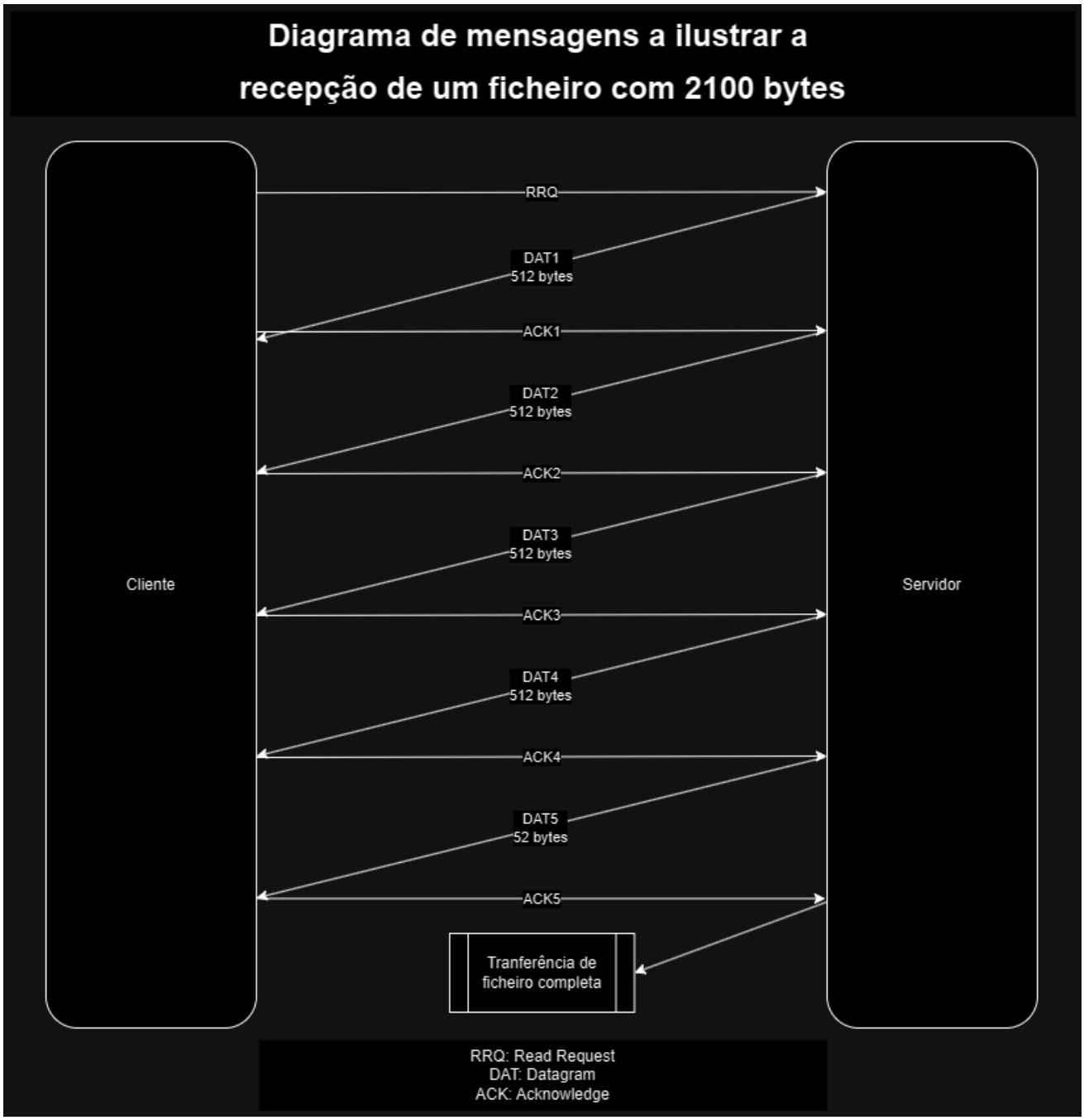


Diagrama de receção de 2100 bytes



Pacote Read Request/Write Request

IP Header	UDP Header	Opcode 1 = RRQ 2 = WRQ	Filename	0	Mode	0
20 bytes	8 bytes	2 bytes	N bytes	1 bit	N bytes	1 bit

Pacote Datagram

Opcode 3 = DAT	Block number	Data
2 bytes	2 bytes	0 - 512 bytes

Pacote Acknowledge

Opcode 4 = ACK	Block number
2 bytes	2 bytes

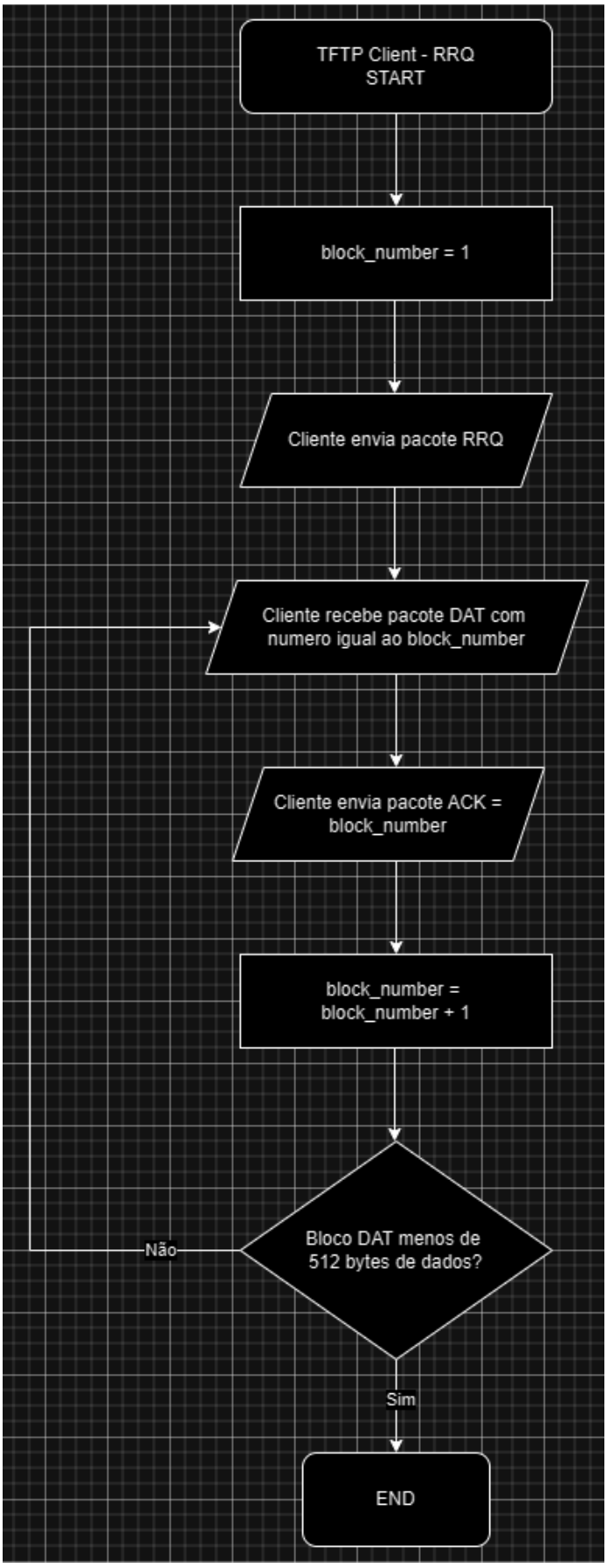
Pacote Error

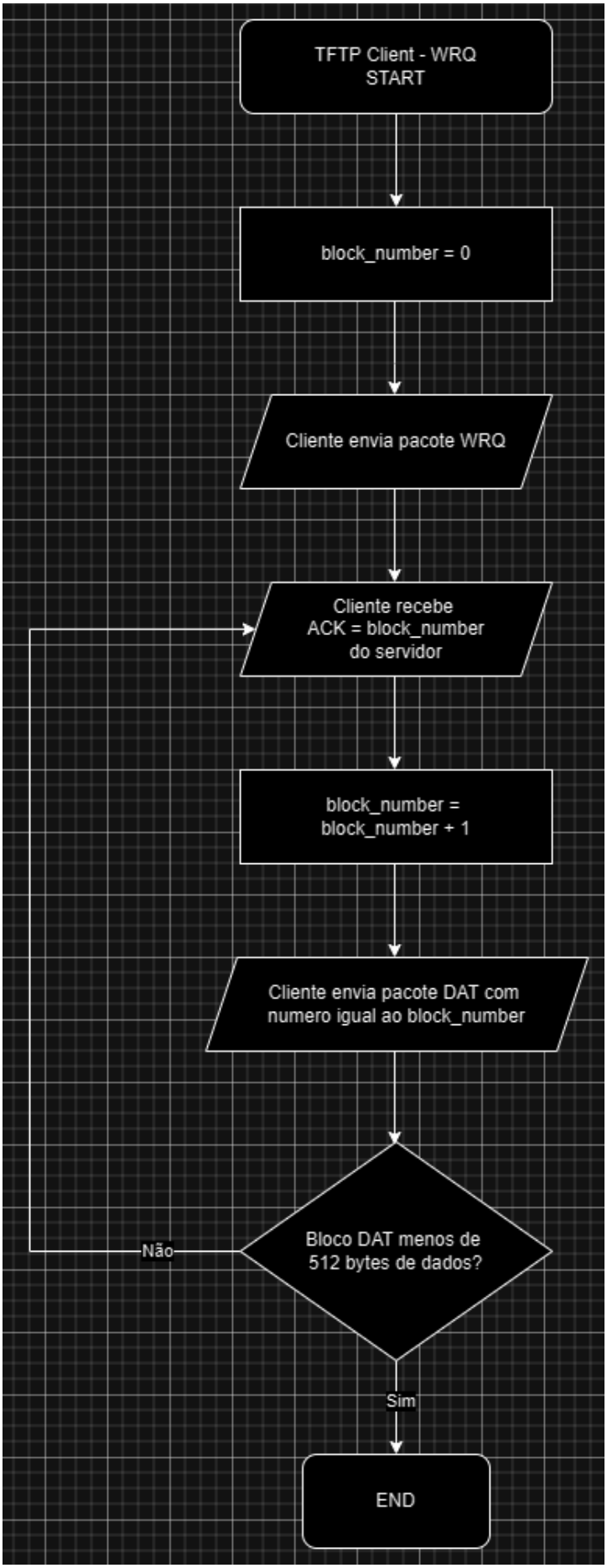
Opcode 5 = ERR	Error number	Error Message	0
2 bytes	2 bytes	N bytes	1 bit

IP Header	Cabeçalho IP, contem todos os dados necessários para o protocolo como a versão (IPv4 ou v6), o protocolo que está a ser utilizado (UDP, TCP, etc.), o checksum para verificar a integridade do cabeçalho, etc.
UDP Header	Cabeçalho UDP, contem todos os dados necessários para o protocolo como o porto de origem e de destino, o checksum para verificar a integridade do cabeçalho e (ao contrário do checksum do IP header) também dos dados.
Opcode	Contem 1 int (16 bits ou 2 bytes) de -32768 a 32767. Neste caso só de 1 a 5, qualquer outro número dá erro.
Filename	O nome do ficheiro, como é uma string pode ter um tamanho variável de N bytes.
Mode	Define o “modo” de transferência. Neste caso o ‘octet’ que transfere os dados em modo binário (raw 8 bit bytes). Mas também existem outros como o netascii que transfere os dados em formato ascii; e mail que manda os dados em formato ascii mas para um user em vez de um ficheiro (embora este modo esteja obsoleto).
0	O bit 0, para terminar os bytes de Filename, Mode, ou Error Message.

Block Number	Contem 1 int (16 bits ou 2 bytes) de -32768 a 32767. Neste caso só números positivos. Por isso o protocolo TFTP não consiga transferir ficheiros muito grandes se não usar o RFC 2348 que introduziu block-size negotiation.
Data	0 – 512 bytes de dados binários que foram packed na origem e seram unpacked no destino. Aceita qualquer tipo de dados.
Error Number	Contem 1 int. Neste caso um número de 0 a 7 dependente do erro definido pelo protocolo TFTP.
Error Message	A mensagem de erro é um string que pode ter um N número de bytes. Por isso tem de ser terminado por um 0.

Desenho e Estrutura




TFTP Client - WRQ
START
block_number = 0
Cliente envia pacote WRQ
Cliente recebe
ACK = block_number
do servidor
block_number =
block_number + 1
Cliente envia pacote DAT com
numero igual ao block_number
Bloco DAT menos de
512 bytes de dados?
Não
Sim
END

Implementação

Os pacotes são gerados usando a biblioteca 'struct' que permite empacotar e desempacotar dados em formato binário. As funções de empacotamento utilizam a função 'struct.pack' para empacotar os dados nos formatos especificados. O resultado é um pacote binário que pode ser enviado através da rede.

O modo interativo do cliente foi implementado maioritariamente com o 'match'. Usando o 'split' para partir o input do user em "cmd" e "*args" deu para fazer 'match' ao 'cmd' e passar os "*args" como argumentos para as funções "get" e "put".

Os temporizadores dos pacotes DAT foram programados utilizando o mecanismo sock.settimeout da biblioteca socket. Depois do envio de um pacote RRQ o cliente espera um determinado tempo (neste caso a variável INACTIVITY_TIMEOUT) e se não receber um pacote dentro desse determinado tempo ele lança uma exceção.

O código foi organizado de maneira simples, o código do cliente está no ficheiro cliente.py: o docopt, o modo não interativo e também o modo interativo. Todas as outras funções (get file, put file, pack/unpack rrq/wrq/dat/ack/err/opcode, errors and exceptions, e common utilities) que possam ser partilhadas entre um cliente e um servidor estão num ficheiro tftp.py.

Ficou implementado o get_file (RRQ), put_file (WRQ), o cliente não interativo e o cliente interativo.
Ficou por implementar o temporizador reenvio ACK/DAT, o servidor, e o comando dir.

No client.py foram usadas as bibliotecas:

- os – para a função clear_screen poder limpar o ecrã dependendo do sistema operativo antes de entrar no modo interativo
- sys – para poder sair do modo interativo com o sys.exit(0)
- subprocess – para poder correr o clear_screen
- textwrap – para de-indentar o texto do comando "help" na forma interativa
- docopt – para o user poder passar os argumentos para o programa e ter documentação sobre como usar o programa
- socket – para ir buscar o IP/nome do servidor quando se usa o modo interativo

No tftp.py foram usadas a bibliotecas:

- os – para fazer 'os.path.getsize' ao "source_file" e poder mostrar quanto de um ficheiro se fez upload
- socket – para toda a connecção e envio de dados
- string – para analisar se o nome do ficheiro é um subset de string.printable
- struct – para fazer pack e unpack aos pacotes
- ipaddress – para encontrar o IP dado um hostname, ou um hostname dado um IP com o ipaddress.ip_address()
- re – para verificar se o hostname dado é valido com o re.compile()
- enum – para importar o IntEnum e criar uma classe de integers para os TFTPOpcodes

Conclusão

O protocolo TFTP, construído em cima do protocolo UDP, é simples e rápido de implementar.

Graças às bibliotecas do Python de ‘struct’ (para transformar ficheiros em binário) e ‘socket’ (para transferir dados entre um host e um server) a parte mais complicada é a própria logica de como (e quando) mandar e receber packets, e quais packets, e em que ordem.

Dito isto, o desenvolvimento do relatório foi fulcral para perceber como o protocolo TFTP funciona e assim como o implementar em código.

Com a implementação do cliente tivemos a oportunidade de usar vários mecanismos do Python, match/case, classes, sockets, structs, os/sys, e também erros e exceções. Assim conseguimos aprofundar a nossa sabedoria do Python e também dos protocolos TFTP e UDP.

Anexo I – UDP

O User Datagram Protocol (UDP) é um protocolo de transporte sem conexão, sem confirmação de entrega e sem controle de fluxo, em contraste com o Transmission Control Protocol (TCP), que é um protocolo de transporte orientado a conexão, com confirmação de entrega e controle de fluxo.

No UDP, os dados são enviados em pacotes (datagrams), que são enviados independentemente uns dos outros, sem garantia de entrega ou ordem. Isso significa que os datagrams podem ser perdidos, duplicados ou entregues em ordem incorreta. Além disso, o UDP não fornece nenhum mecanismo de controle de fluxo, o que pode levar a perda de pacotes se a taxa de envio exceder a capacidade da rede.

UDP é mais simples e eficiente que o TCP o que significa que o UDP é menos confiável do que o TCP, pois não garante a entrega ou a ordem dos dados. Portanto, o UDP é geralmente usado em aplicativos que podem tolerar a perda de dados ou que requerem baixa latência, como streaming de áudio ou vídeo.

Webgrafia

Investigação:

- Trivial File Transfer Protocol - Wikipedia
- What is TFTP (Trivial File Transfer Protocol)? - GeeksforGeeks
- TFTP: What is the Trivial File Transfer Protocol all about? - IONOS

Programação:

- docopt—language for description of command-line interfaces
- os — Miscellaneous operating system interfaces — Python 3.12.2 documentation
- socket — Low-level networking interface — Python 3.12.2 documentation
- struct — Interpret bytes as packed binary data — Python 3.12.2 documentation
- sys — System-specific parameters and functions — Python 3.12.2 documentation
- subprocess — Subprocess management — Python 3.12.2 documentation
- 8. Errors and Exceptions — Python 3.12.2 documentation
- 9. Classes — Python 3.12.2 documentation

Anexo I:

- User Datagram Protocol - Wikipedia
- User Datagram Protocol (UDP) - GeeksforGeeks
- What is the User Datagram Protocol (UDP)? | Cloudflare

Fluxogramas:

- draw.io (diagrams.net)