# PRÀCTICA 2

Rems Nalivaiko

Pol Rubio

05 de novembre de 2023

103311 - XARXES I SERVEIS

Fundació TecnoCampus Mataró-Maresme

# Index

# Introduction

This Python project is designed to function as a TFTP (Trivial File Transfer Protocol) client, enabling users to upload and download files from a TFTP server. The client can interact with a TFTP server to facilitate file transfers using the TFTP protocol defined in the RFC1350.

The primary objective of this project is to implement a TFTP client that can perform file upload (put) and file download (get) operations efficiently. It uses the TFTP protocol to send and receive data packets to and from the server, following the TFTP specification

In the following sections, we will delve into the details of how the program operates, explaining the key components and functions involved in achieving its goals. This documentation will guide you through the core classes and methods, providing a clear understanding of how the TFTP client was designed, implemented, and how to utilise it effectively

# Program Execution

The Python script developed for this project is a TFTP (Trivial File Transfer Protocol) client. It allows users to upload or download files from a TFTP server. This section explains the script's functionality, the structure of the TFTPController class, and how it facilitates file transfer

## Script Overview

The script is structured around the TFTPController class, which is responsible for handling TFTP requests and managing file transfers

### TFTPController Class:

The TFTPController class is the core component of this script, responsible for interacting with the TFTP server. It provides methods for sending read and write requests, as well as functions for sending and receiving data packets. The class also handles error conditions, timeouts, and other aspects of TFTP communication

The TFTPController class uses UDP sockets for communication with the TFTP server and can perform two main actions: 'put' (upload a file to the server) and 'get' (download a file from the server). It takes user-provided arguments such as the action type ('put' or 'get'), the filename, server IP or hostname, and additional options like the port number and socket timeout

The script employs the TFTP protocol's specific opcodes, packet structures, and error handling mechanisms to ensure reliable file transfers between the client and the TFTP server

# Execution Flow

## 1. Argument Parsing:

The script starts by parsing command-line arguments using the argparse module. It retrieves user inputs such as the action, the filename to upload or download, the TFTP server's IP address or hostname

## 2. TFTP Controller Initialization:

An instance of the `TFTPController` class is created, initializing the UDP socket for communication with the TFTP server and setting up logging for the script

## 3. File Transfer Action:

Depending on the specified action ('get' or 'put'), the script performs one of two actions:

- If the action is 'put', the `put` method is called to upload the specified file to the TFTP server
- If the action is 'get', the `get` method is called to download a file from the TFTP server

## 4. Error Handling:

The script includes error handling to catch and handle exceptions that may occur during file transfer operations.

## 5. Clean-up:

Finally, the UDP socket is closed to release system resources.

# Prerequisites

- Python 3.8 or above.
- Required libraries: `argparse`, `socket`.
- A file to transfer.
- Access to a TFTP server.

# Functionality Explanation

In the provided Python code, we have a TFTP (Trivial File Transfer Protocol) client implemented, which allows you to perform file uploads ("put") and downloads ("get") with a TFTP server. Below is an adaptation of the example text to the code's functionalities:

## TFTPController Class

The `TFTPController` class serves as the core of the TFTP client, encapsulating the TFTP protocol operations. It handles sending requests, receiving data, and managing the transfer process.

### Class Constant Attributes:

`TFPT_OPCODES`: A dictionary that maps TFTP operation codes to their corresponding values.
`HEADER_SIZE`: The size of the TFTP packet header in bytes.
`DATA_SIZE`: The maximum size of data that can be sent in a single TFTP packet.
`MAX_RETRIES`: The maximum number of retries for a packet.
`TIMEOUT`: The socket timeout in seconds.

### Packet Class:

The `Packet` class defines three subclasses, including `Error`, `Ack`, and `Data`, representing different types of TFTP packets. These classes are used to transform binary data received from the server into structured packet objects

### Constructor __init__(self, server_ip: str, port: int, socket_timeout: float = TIMEOUT)`:

Initialises the TFTP controller, creating a UDP socket for communication with the TFTP server. It takes the server's IP address, port, and an optional socket timeout as arguments.

**Methods**:

```
send_request()
```

Sends a TFTP request packet (either Read Request-RRQ or Write Request-WRQ) to the server with the specified opcode, filename, and mode.

```
send_data(self, block: int, data: bytes)
```

Sends a TFTP Data packet to the server with the specified block number and data.

```
send_ack(self, block: int)
```

Sends a TFTP Acknowledgement packet to the server with the specified block number.

```
receive_data(self)
```

Waits for a TFTP packet from the server and returns the received data and the server's address.

```
transform_data(self, data: bytes)
```

Transforms binary data into a structured TFTP packet object, identifying the packet type (Error, Ack, or Data).

```
listen_packet(self)
```

Waits for a UDP answer from the server and returns a packet object.

```
split_file(self, local_filename: str)
```

Reads and splits a local file into chunks, yielding data for transmission. Handles exceptions for file access.

```
expect_packet(self, block_number: int, packet_type)
```

Expects to receive a specific type of TFTP packet with the given block number, raising exceptions for errors or unexpected packets.

```
put(self, local_filename: str)
```

Uploads a file to the TFTP server, sending Data packets and handling Acknowledgement packets.

`get(self, remote_filename: str)`

Downloads a file from the TFTP server, receiving Data packets and sending Acknowledgment packets.

`close(self)`

Closes the UDP socket used for communication.

## Execution entry

The script execution begins here, where command-line arguments are parsed, and an instance of the `` `TFTPController` `` class is created to perform TFTP operations for file upload and download.

### Argument Parsing:

Utilises the `argparse` library to define, parse, and handle command-line arguments, enabling user input for various parameters:

1. `action`**:** This is the first argument and represents the action to be performed. Users can choose between 'get' to download a file and 'put' to upload a file.
2. `filename`**:** This argument specifies the name of the file to upload or download.
3. `server`**:** The IP address or hostname of the TFTP server to connect to.
4. `-p` **or** `--port`**:** An optional argument for specifying the port number of the TFTP server. If not provided, it defaults to 69.
5. `-t` **or** `--timeout`**:** An optional argument to set the socket timeout in seconds. If not provided, it defaults to 1 second.

Users can run the script from the command line and provide these arguments to configure the behaviour of the TFTP client. For example, they can specify the action ('get' or 'put'), the filename, the TFTP server's address, the port, and the socket timeout. This allows the script to be versatile and adaptable for different TFTP operations.

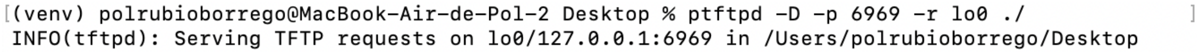
# Example of execution 1

1. First, we need to install the ptftpd package list with the command:

```
pip install ptftpd
```

2. Start the TFTP server in the lo0 interface and in the 6969 port.

```
ptftpd -D -p 6969 -r lo0 ./
```

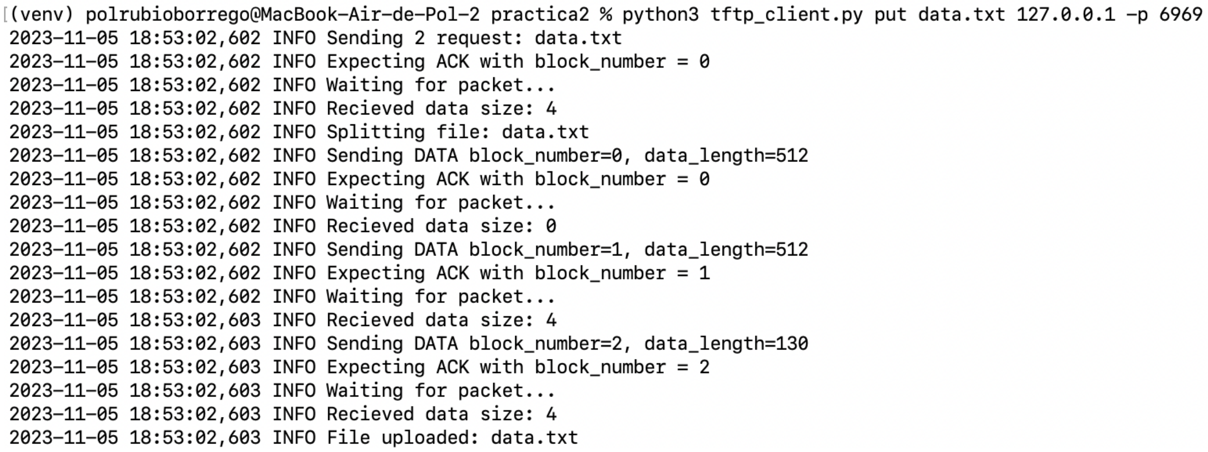


```
[(venv) polrubioborrego@MacBook-Air-de-Pol-2 Desktop % ptftpd -D -p 6969 -r lo0 ./            ]
 INFO(tftpd): Serving TFTP requests on lo0/127.0.0.1:6969 in /Users/polrubioborrego/Desktop
```

Figure 1: Terminal message of the TFTP server started

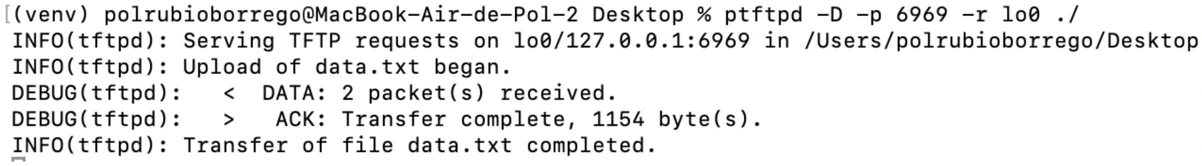
3. Run the code of our program with the following command, in this example we will put the file "data.txt" in the local IP (127.0.0.1) and the port 6969, and leave all the other arguments in default

```
python3 tftp_client.py put data.txt 127.0.0.1
-p 6969
```



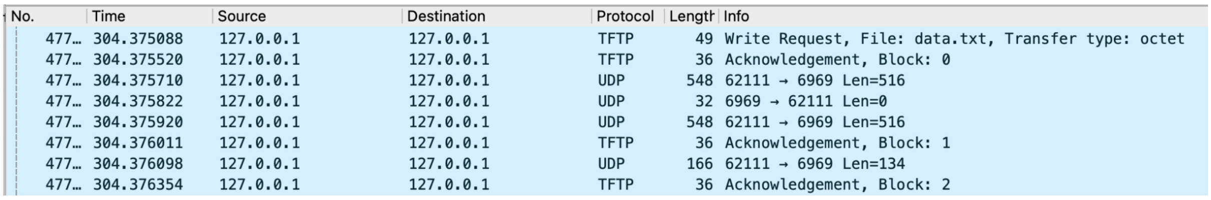
```
[(venv) polrubioborrego@MacBook-Air-de-Pol-2 practica2 % python3 tftp_client.py put data.txt 127.0.0.1 -p 6969
 2023-11-05 18:53:02,602 INFO Sending 2 request: data.txt
 2023-11-05 18:53:02,602 INFO Expecting ACK with block_number = 0
 2023-11-05 18:53:02,602 INFO Waiting for packet...
 2023-11-05 18:53:02,602 INFO Recieved data size: 4
 2023-11-05 18:53:02,602 INFO Splitting file: data.txt
 2023-11-05 18:53:02,602 INFO Sending DATA block_number=0, data_length=512
 2023-11-05 18:53:02,602 INFO Expecting ACK with block_number = 0
 2023-11-05 18:53:02,602 INFO Waiting for packet...
 2023-11-05 18:53:02,602 INFO Recieved data size: 0
 2023-11-05 18:53:02,602 INFO Sending DATA block_number=1, data_length=512
 2023-11-05 18:53:02,602 INFO Expecting ACK with block_number = 1
 2023-11-05 18:53:02,602 INFO Waiting for packet...
 2023-11-05 18:53:02,603 INFO Recieved data size: 4
 2023-11-05 18:53:02,603 INFO Sending DATA block_number=2, data_length=130
 2023-11-05 18:53:02,603 INFO Expecting ACK with block_number = 2
 2023-11-05 18:53:02,603 INFO Waiting for packet...
 2023-11-05 18:53:02,603 INFO Recieved data size: 4
 2023-11-05 18:53:02,603 INFO File uploaded: data.txt
```

Figure 2: Terminal messages of the python program execution (client side)



```
(venv) polrubioborrego@MacBook-Air-de-Pol-2 Desktop % ptftpd -D -p 6969 -r lo0 ./
INFO(tftpd): Serving TFTP requests on lo0/127.0.0.1:6969 in /Users/polrubioborrego/Desktop
INFO(tftpd): Upload of data.txt began.
DEBUG(tftpd):    <  DATA: 2 packet(s) received.
DEBUG(tftpd):    >   ACK: Transfer complete, 1154 byte(s).
INFO(tftpd): Transfer of file data.txt completed.
```

Figure 3: Terminal messages of the TFTP program (server side)



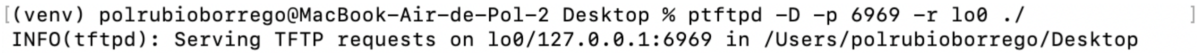
| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 477… | 304.375088 | 127.0.0.1 | 127.0.0.1 | TFTP | 49 | Write Request, File: data.txt, Transfer type: octet |
| 477… | 304.375520 | 127.0.0.1 | 127.0.0.1 | TFTP | 36 | Acknowledgement, Block: 0 |
| 477… | 304.375710 | 127.0.0.1 | 127.0.0.1 | UDP | 548 | 62111 → 6969 Len=516 |
| 477… | 304.375822 | 127.0.0.1 | 127.0.0.1 | UDP | 32 | 6969 → 62111 Len=0 |
| 477… | 304.375920 | 127.0.0.1 | 127.0.0.1 | UDP | 548 | 62111 → 6969 Len=516 |
| 477… | 304.376011 | 127.0.0.1 | 127.0.0.1 | TFTP | 36 | Acknowledgement, Block: 1 |
| 477… | 304.376098 | 127.0.0.1 | 127.0.0.1 | UDP | 166 | 62111 → 6969 Len=134 |
| 477… | 304.376354 | 127.0.0.1 | 127.0.0.1 | TFTP | 36 | Acknowledgement, Block: 2 |

Figure 4: Wireshark analysis of the transaction

# Example of execution 2

1. Start the TFTP server in the lo0 interface and in the 6969 port.

```
ptftpd -D -p 6969 -r lo0 ./
```

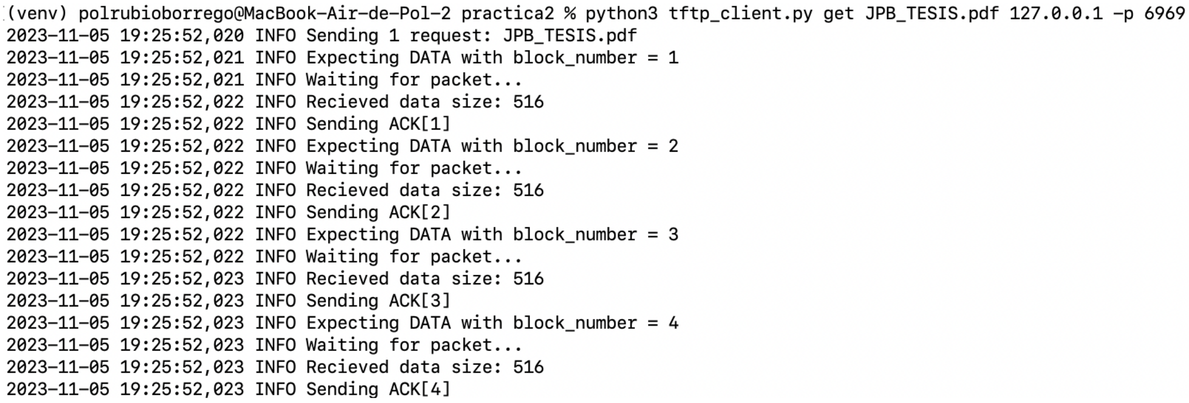


```
(venv) polrubioborrego@MacBook-Air-de-Pol-2 Desktop % ptftpd -D -p 6969 -r lo0 ./
INFO(tftpd): Serving TFTP requests on lo0/127.0.0.1:6969 in /Users/polrubioborrego/Desktop
```

Figure 5: Terminal message of the TFTP server started

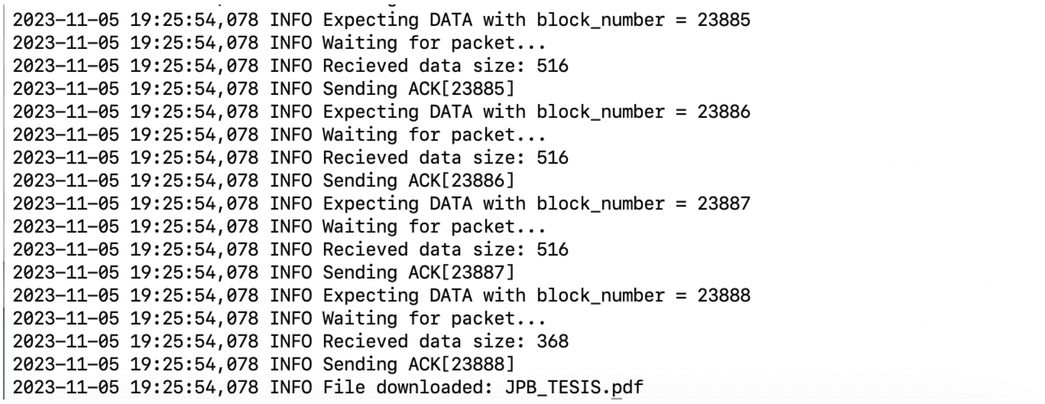
2. Run the code of our program with the following command, in this example we will get the file “JPB_TESIS.pdf” from the local ip (127.0.0.1) and the port 6969, and leave all the other arguments in default

```
python3 tftp_client.py get JPB_TESIS.pdf
127.0.0.1 -p 6969
```



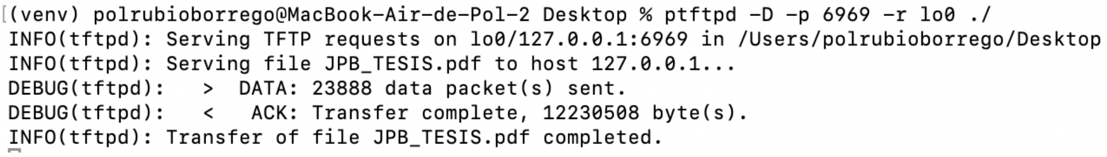
```
(venv) polrubioborrego@MacBook-Air-de-Pol-2 practica2 % python3 tftp_client.py get JPB_TESIS.pdf 127.0.0.1 -p 6969
2023-11-05 19:25:52,020 INFO Sending 1 request: JPB_TESIS.pdf
2023-11-05 19:25:52,021 INFO Expecting DATA with block_number = 1
2023-11-05 19:25:52,021 INFO Waiting for packet...
2023-11-05 19:25:52,022 INFO Recieved data size: 516
2023-11-05 19:25:52,022 INFO Sending ACK[1]
2023-11-05 19:25:52,022 INFO Expecting DATA with block_number = 2
2023-11-05 19:25:52,022 INFO Waiting for packet...
2023-11-05 19:25:52,022 INFO Recieved data size: 516
2023-11-05 19:25:52,022 INFO Sending ACK[2]
2023-11-05 19:25:52,022 INFO Expecting DATA with block_number = 3
2023-11-05 19:25:52,022 INFO Waiting for packet...
2023-11-05 19:25:52,023 INFO Recieved data size: 516
2023-11-05 19:25:52,023 INFO Sending ACK[3]
2023-11-05 19:25:52,023 INFO Expecting DATA with block_number = 4
2023-11-05 19:25:52,023 INFO Waiting for packet...
2023-11-05 19:25:52,023 INFO Recieved data size: 516
2023-11-05 19:25:52,023 INFO Sending ACK[4]
```

Figure 6: Terminal messages of the first 4 blocks from the python program execution (client side)



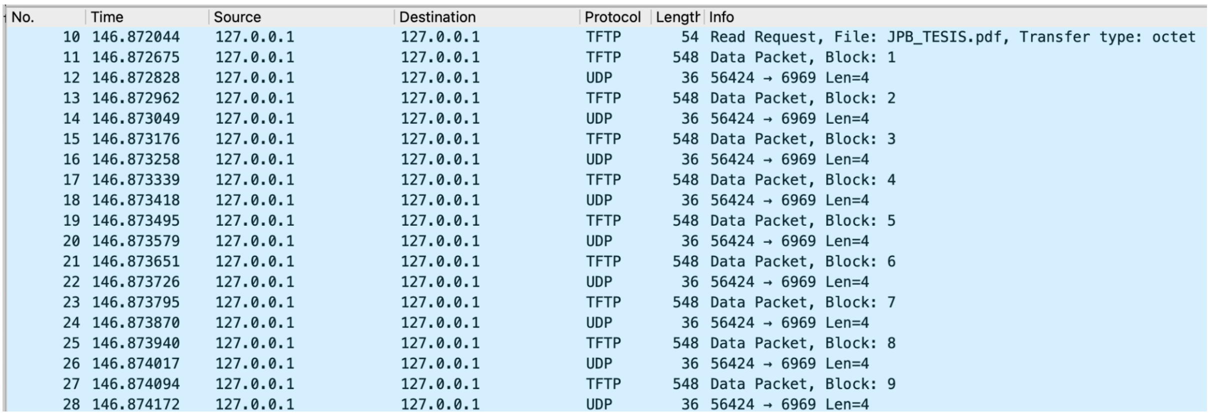
```
2023-11-05 19:25:54,078 INFO Expecting DATA with block_number = 23885
2023-11-05 19:25:54,078 INFO Waiting for packet...
2023-11-05 19:25:54,078 INFO Recieved data size: 516
2023-11-05 19:25:54,078 INFO Sending ACK[23885]
2023-11-05 19:25:54,078 INFO Expecting DATA with block_number = 23886
2023-11-05 19:25:54,078 INFO Waiting for packet...
2023-11-05 19:25:54,078 INFO Recieved data size: 516
2023-11-05 19:25:54,078 INFO Sending ACK[23886]
2023-11-05 19:25:54,078 INFO Expecting DATA with block_number = 23887
2023-11-05 19:25:54,078 INFO Waiting for packet...
2023-11-05 19:25:54,078 INFO Recieved data size: 516
2023-11-05 19:25:54,078 INFO Sending ACK[23887]
2023-11-05 19:25:54,078 INFO Expecting DATA with block_number = 23888
2023-11-05 19:25:54,078 INFO Waiting for packet...
2023-11-05 19:25:54,078 INFO Recieved data size: 368
2023-11-05 19:25:54,078 INFO Sending ACK[23888]
2023-11-05 19:25:54,078 INFO File downloaded: JPB_TESIS.pdf
```

Figure 7: Terminal messages of the last 4 blocks from the python program execution (client side)



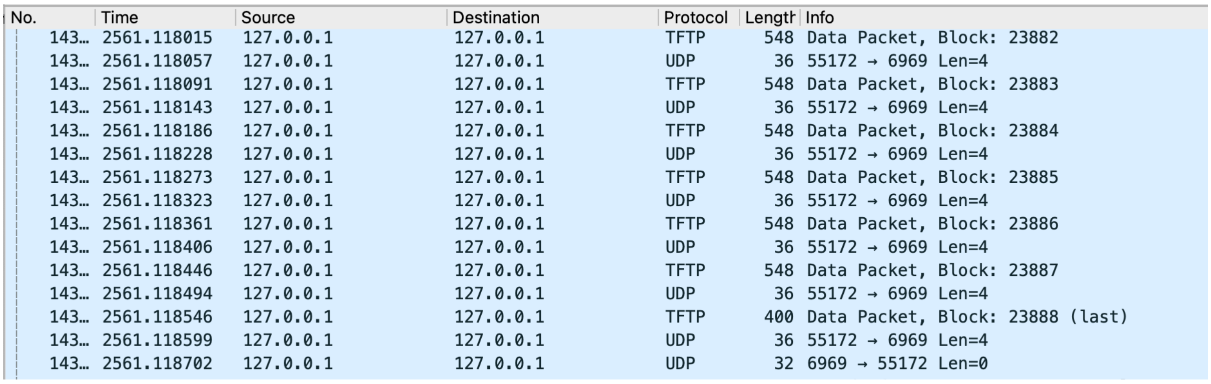
```
(venv) polrubioborrego@MacBook-Air-de-Pol-2 Desktop % ptftpd -D -p 6969 -r lo0 ./
INFO(tftpd): Serving TFTP requests on lo0/127.0.0.1:6969 in /Users/polrubioborrego/Desktop
INFO(tftpd): Serving file JPB_TESIS.pdf to host 127.0.0.1...
DEBUG(tftpd):   >  DATA: 23888 data packet(s) sent.
DEBUG(tftpd):   <   ACK: Transfer complete, 12230508 byte(s).
INFO(tftpd): Transfer of file JPB_TESIS.pdf completed.
```

Figure 8: Terminal messages of the TFTP program (server side)



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 10 | 146.872044 | 127.0.0.1 | 127.0.0.1 | TFTP | 54 | Read Request, File: JPB_TESIS.pdf, Transfer type: octet |
| 11 | 146.872675 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 1 |
| 12 | 146.872828 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 56424 → 6969 Len=4 |
| 13 | 146.872962 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 2 |
| 14 | 146.873049 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 56424 → 6969 Len=4 |
| 15 | 146.873176 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 3 |
| 16 | 146.873258 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 56424 → 6969 Len=4 |
| 17 | 146.873339 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 4 |
| 18 | 146.873418 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 56424 → 6969 Len=4 |
| 19 | 146.873495 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 5 |
| 20 | 146.873579 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 56424 → 6969 Len=4 |
| 21 | 146.873651 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 6 |
| 22 | 146.873726 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 56424 → 6969 Len=4 |
| 23 | 146.873795 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 7 |
| 24 | 146.873870 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 56424 → 6969 Len=4 |
| 25 | 146.873940 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 8 |
| 26 | 146.874017 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 56424 → 6969 Len=4 |
| 27 | 146.874094 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 9 |
| 28 | 146.874172 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 56424 → 6969 Len=4 |

Figure 9: Wireshark analysis of the transaction (first 9 blocks)



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 143… | 2561.118015 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 23882 |
| 143… | 2561.118057 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 55172 → 6969 Len=4 |
| 143… | 2561.118091 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 23883 |
| 143… | 2561.118143 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 55172 → 6969 Len=4 |
| 143… | 2561.118186 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 23884 |
| 143… | 2561.118228 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 55172 → 6969 Len=4 |
| 143… | 2561.118273 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 23885 |
| 143… | 2561.118323 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 55172 → 6969 Len=4 |
| 143… | 2561.118361 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 23886 |
| 143… | 2561.118406 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 55172 → 6969 Len=4 |
| 143… | 2561.118446 | 127.0.0.1 | 127.0.0.1 | TFTP | 548 | Data Packet, Block: 23887 |
| 143… | 2561.118494 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 55172 → 6969 Len=4 |
| 143… | 2561.118546 | 127.0.0.1 | 127.0.0.1 | TFTP | 400 | Data Packet, Block: 23888 (last) |
| 143… | 2561.118599 | 127.0.0.1 | 127.0.0.1 | UDP | 36 | 55172 → 6969 Len=4 |
| 143… | 2561.118702 | 127.0.0.1 | 127.0.0.1 | UDP | 32 | 6969 → 55172 Len=0 |

Figure 10: Wireshark analysis of the transaction (last 7 blocks)

# Qüestions d’implementació

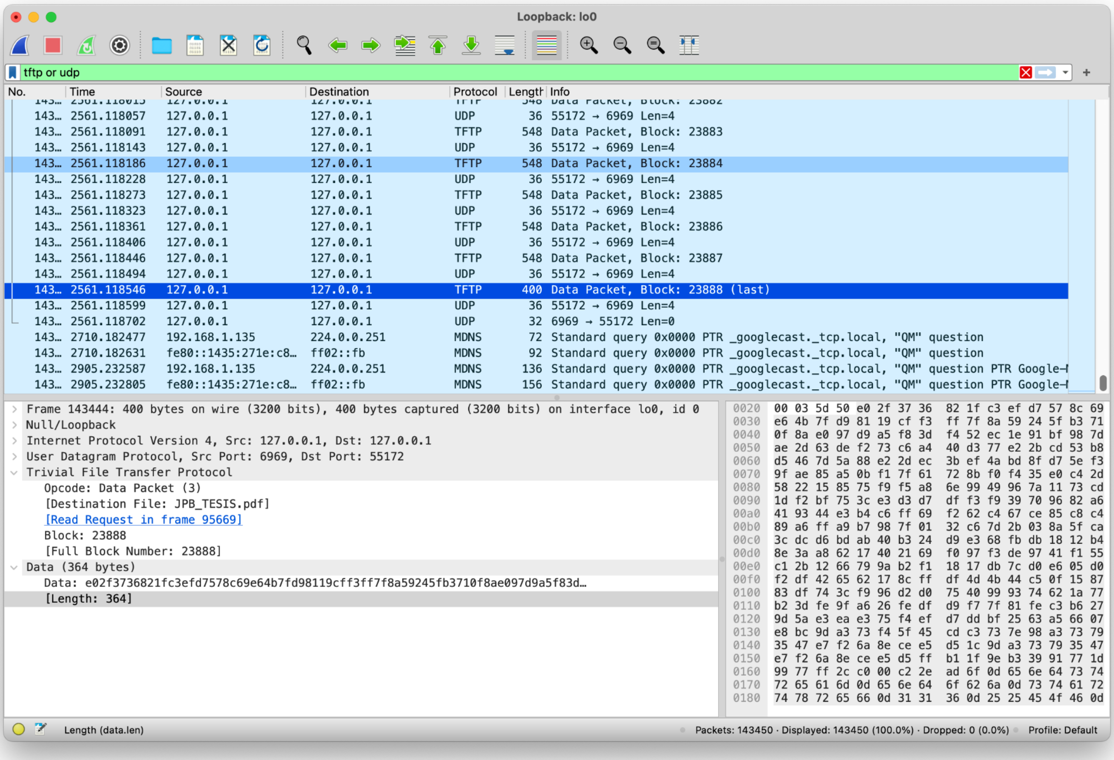
1. **Quina és la mida del camp Opcode la capçalera TFTP, quins són els tipus de codis d’operació del protocol TFTP i quins valors tenen al camp Opcode?**
    a. 2 bytes
    b. Els valors possibles són:
        1. Read request (RRQ)
        2. Write request (WRQ)
        3. Data (DATA)
        4. Acknowledgment (ACK)
        5. Error (ERROR)

2. **Quina és la mida del camp ErrorCode, quins són els tipus de codis d’error del protocol TFTP i quins valors tenen al camp ErrorCode?**
    a. 2 bytes
    b. El valors possibles són:
        0. Not defined, see error message (if any).
        1. File not found.
        2. Access violation.
        3. Disk full or allocation exceeded.
        4. Illegal TFTP operation.
        5. Unknown transfer ID.
        6. File already exists.
        7. No such user.

3. **Com s’indica la fi d’una cadena de caràcters de mida variable en la capçalera TFTP?**
    a. “\n”

4. **Quin és el protocol de la capa de transport que utilitza el protocol TFTP i en quin port escolta per defecte el servidor?**
    a. UDP
    b. port 69

5. **Quina és la condició que permet a un client o servidor TFTP detectar la fi de la transferència d’un fitxer? Mostra una captura de Wireshark on s’observi.**
    a. La fi de la transferència d’un fitxer és detectada quan la data d’un packet té una longitud inferior a 512 bytes



tftp or udp

```
No.   Time         Source               Destination    Protocol Length Info
143…  2561.118015  127.0.0.1            127.0.0.1      TFTP     548    Data Packet, Block: 23882
143…  2561.118057  127.0.0.1            127.0.0.1      UDP       36    55172 → 6969 Len=4
143…  2561.118091  127.0.0.1            127.0.0.1      TFTP     548    Data Packet, Block: 23883
143…  2561.118143  127.0.0.1            127.0.0.1      UDP       36    55172 → 6969 Len=4
143…  2561.118186  127.0.0.1            127.0.0.1      TFTP     548    Data Packet, Block: 23884
143…  2561.118228  127.0.0.1            127.0.0.1      UDP       36    55172 → 6969 Len=4
143…  2561.118273  127.0.0.1            127.0.0.1      TFTP     548    Data Packet, Block: 23885
143…  2561.118323  127.0.0.1            127.0.0.1      UDP       36    55172 → 6969 Len=4
143…  2561.118361  127.0.0.1            127.0.0.1      TFTP     548    Data Packet, Block: 23886
143…  2561.118406  127.0.0.1            127.0.0.1      UDP       36    55172 → 6969 Len=4
143…  2561.118446  127.0.0.1            127.0.0.1      TFTP     548    Data Packet, Block: 23887
143…  2561.118494  127.0.0.1            127.0.0.1      UDP       36    55172 → 6969 Len=4
143…  2561.118546  127.0.0.1            127.0.0.1      TFTP     400    Data Packet, Block: 23888 (last)
143…  2561.118599  127.0.0.1            127.0.0.1      UDP       36    55172 → 6969 Len=4
143…  2561.118702  127.0.0.1            127.0.0.1      UDP       32    6969 → 55172 Len=0
143…  2710.182477  192.168.1.135        224.0.0.251    MDNS      72    Standard query 0x0000 PTR _googlecast._tcp.local, "QM" question
143…  2710.182631  fe80::1435:271e:c8…  ff02::fb       MDNS      92    Standard query 0x0000 PTR _googlecast._tcp.local, "QM" question
143…  2905.232587  192.168.1.135        224.0.0.251    MDNS     136    Standard query 0x0000 PTR _googlecast._tcp.local, "QM" question PTR Google-I
143…  2905.232805  fe80::1435:271e:c8…  ff02::fb       MDNS     156    Standard query 0x0000 PTR _googlecast._tcp.local, "QM" question PTR Google-I
```

```
> Frame 143444: 400 bytes on wire (3200 bits), 400 bytes captured (3200 bits) on interface lo0, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 6969, Dst Port: 55172
v Trivial File Transfer Protocol
    Opcode: Data Packet (3)
    [Destination File: JPB_TESIS.pdf]
    [Read Request in frame 95669]
    Block: 23888
    [Full Block Number: 23888]
v Data (364 bytes)
    Data: e02f3736821fc3efd7578c69e64b7fd98119cff3ff7f8a59245fb3710f8ae097d9a5f83d…
    [Length: 364]
```

```
0020  00 03 5d 50 e0 2f 37 36  82 1f c3 ef d7 57 8c 69
0030  e6 4b 7f d9 81 19 cf f3  ff 7f 8a 59 24 5f b3 71
0040  0f 8a e0 97 d9 a5 f8 3d  f4 52 ec 1e 91 bf 98 7d
0050  ae 2d 63 de f2 73 c6 a4  40 d3 77 e2 2b cd 53 b8
0060  d5 46 7d 5a 88 e2 2d ec  3b ef 4a bd 8f d7 5e f3
0070  9f ae 85 a5 0b f1 7f 61  72 8b f0 f4 35 e0 c4 2d
0080  58 22 15 85 75 f9 f5 a8  6e 99 49 96 7a 11 73 cd
0090  1d f2 bf 75 3c e3 d3 d7  df f3 f9 39 70 96 82 a6
00a0  41 93 44 e3 b4 c6 ff 69  f2 62 c4 67 ce 85 c8 c4
00b0  89 a6 ff a9 b7 98 7f 01  32 c6 7d 2b 03 8a 5f ca
00c0  3c dc d6 bd ab 40 b3 24  d9 e3 68 fb db 18 12 b4
00d0  8e 3a a8 62 17 40 21 69  f0 97 f3 de 97 41 f1 55
00e0  c1 2b 12 66 79 9a b2 f1  18 17 db 7c d0 e6 05 d0
00f0  f2 df 42 65 62 17 8c ff  df 4d 4b 44 c5 0f 15 87
0100  83 df 74 3c f9 96 d2 d0  75 40 99 93 74 62 1a 77
0110  b2 3d fe 9f a6 26 fe df  d9 f7 7f 81 fe c3 b6 27
0120  9d 5a e3 ea e3 75 f4 ef  d7 dd bf 25 63 a5 66 07
0130  e8 bc 9d a3 73 f4 5f 45  cd c3 73 7e 98 a3 73 79
0140  35 47 e7 f2 6a 8e ce e5  d5 1c 9d a3 73 79 35 47
0150  e7 f2 6a 8e ce e5 d5 ff  b1 1f 9e b3 39 91 77 1d
0160  99 77 ff 2c c0 00 c2 2e  ad 6f 0d 65 6e 64 73 74
0170  72 65 61 6d 0d 65 6e 64  6f 62 6a 0d 73 74 61 72
0180  74 78 72 65 66 0d 31 31  36 0d 25 25 45 4f 46 0d
```

Length (data.len)

Packets: 143450 · Displayed: 143450 (100.0%) · Dropped: 0 (0.0%) · Profile: Default

Figure 11: Wireshark analysis of the transaction (last 6 blocks and details of the last one)

# Bibliography

**Python Standard Library:**

**TFTP Protocol:**

RFC 1350: https://tools.ietf.org/html/rfc1350

**ptftp:**

Official documentation:

**Python Standard Library:**

Official documentation: https://docs.python.org/3/library/index.html

**socket:**

Official documentation: https://docs.python.org/3/library/socket.html

**argparse:**

Official documentation: https://docs.python.org/3/library/argparse.html

**logging:**

Official documentation: https://docs.python.org/3/library/logging.html

**struct:**

Official documentation: https://docs.python.org/3/library/struct.html

**os.path:**

Official documentation: https://docs.python.org/3/library/os.path.html