BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

TFTP CLIENT AND SERVER

TFTP KLIENT A SERVER

TERM PROJECT
SEMESTRÁLNÍ PROJEKT

AUTHOR **ŠTEFAN PEKNÍK**
AUTOR PRÁCE

SUPERVISOR **Ing. DANIEL DOLEJŠKA**
VEDOUCÍ PRÁCE

BRNO 2023

Contents

1 Introduction **2**

2 Trivial File Transfer Protocol **3**
2.1 Default TFTP . . . 3
2.1.1 Packets . . . 3
2.1.2 Communication . . . 3
2.2 Options Extension . . . 4
2.2.1 Changes to default communication . . . 4
2.2.2 Blocksize Option . . . 4
2.2.3 Timeout Option . . . 5
2.2.4 Transfer Size option . . . 5

3 Implementation **6**
3.1 Common parts . . . 6
3.2 TFTP Client . . . 8
3.2.1 Graph Of Dependencies . . . 8
3.2.2 How to Use . . . 8
3.3 TFTP Server . . . 9
3.3.1 Graph Of Dependencies . . . 9
3.3.2 How To Use . . . 9
3.4 Known limitations . . . 10

Bibliography **11**

Chapter 1

Introduction

This document serves as a comprehensive guide to the design, implementation, and usage of a Trivial File Transfer Protocol (TFTP) client and server. The TFTP protocol, specified in RFC 1350[5], provides a simple and lightweight method for transferring files between devices over a network. This documentation aims to provide a clear understanding of the TFTP client and server developed as part of this project.

Chapter 2

Trivial File Transfer Protocol

2.1 Default TFTP

TFTP (Trivial File Transfer Protocol) is a very simple protocol used to transfer files.

It was designed as simple and lightweight, so it lacks a lot of features of regular FTP, such as authentication. The only thing it can do is read and write files from/to a remote server.

2.1.1 Packets

The transfer is done between clients and a server using UDP communication to transfer TFTP packets. Each packet is acknowledged by the opposite side to signal that the packet was successfully received. There are five types of TFTP packets:

- WRQ - write request;
- RRQ - read request;
- DATA - data;
- ACK - acknowledgment;
- ERROR - error.

2.1.2 Communication

Each and every communication is started by a client by creating and sending a WRQ (as a request to write into a server) or a RRQ packet (as a request to read from a server) to a server.

If the server receives a WRQ packet, it responds with ACK packet (block number set to zero). The client then proceeds with sending first DATA packet (block number set to one). The server acknowledges each received DATA packet by responding with an ACK packet with block number set to the block number in the received DATA packet. This goes on (incrementing block number by one each time) until less then 512 bytes of data are sent in a DATA packet, which signals an end of the file. The server receives last DATA packet and ends communication with the client. The server can but is not obligated to respond with an ACK to the last DATA packet. The client tries to receive the last ACK packet, but even if no ACK comes, the client ends successfully.

If the server receives a RRQ packet, it responds with first DATA packet (block number set to 1). The client receives first DATA packet and responds with an ACK packet. This goes on (incrementing block number by one each time) until less then 512 bytes of data are sent in a DATA packet, which signals an end of the file. The server receives last DATA packet and ends communication with the client. The client can but is not obligated to respond with an ACK to the last DATA packet. The server tries to receive the last ACK packet, but even if no ACK comes, the server considers the communication as success.

At any point the client or the server can send or receive an ERROR packet, signaling a problem (can be internal or triggered by a mistake in the communication). The ERROR packet results in an unsuccessful end of the transfer.

More details (including formats of each packet and more) can be found in RFC 1350[5].

2.2 Options Extension

As described in RFC 2347[2], options are a form to modify the default behaviour of TFTP. It enables a client to propose specific options for the file transfer which a server can either accept, modify (only specific ones) or ignore.

Our client and our server will support these options:

- Blocksize option,
- Timeout option,
- Transfer Size option.

2.2.1 Changes to default communication

To handle the newly introduced negotiation of options, a new TFTP packet is introduced - OACK (option acknowledgement) packet and a new error with error number 8 - terminate transfer due to option negotiation.

A client proposes its options and its desired values as part of RRQ or WRQ. If a server receives a RRQ packet with options, it handles those options and responds with OACK containing accepted options with accepted (in some cases potentially modified) values. If the client accepts the negotiated options, it responds with an ACK with block number set to zero. Server then proceeds with standard communication by sending first DATA packet. If a server receives a WRQ packet with options, it handles those options and responds OACK containing accepted options with accepted (in some cases potentially modified) values. If the client accepts the negotiated options, it responds with first DATA packet and default communication proceeds. The client can also not accept the returned options and its values by responding with the new error and ending the transfer.

More can be found in RFC 2347[2].

2.2.2 Blocksize Option

Blocksize option enables to optimize the size of data transferred in each DATA packet. It changes the default maximum size of data in DATA packet to its defined value, which has to be in range between „8“ and „65464“. A server can negotiate only an equal or lower value then the one proposed by a client.

Proof of concept and more in RFC 2348[4].

2.2.3 Timeout Option

Timeout option enables to agree on a specific number of seconds which will both client and server wait for incoming response before re-transmitting their last response. This makes client and server implementation more unified.

Specific details specified in RFC 2349[3].

2.2.4 Transfer Size option

Transfer Size option gives both client and server chance to share information about the size of data to be transported and accordingly react before trying to receive all the data. This is useful when one party is supposed to write down the transported data but at one point realize that there is no more free space. At this point we could have already transported large number of data, but are not able to continue. Transfer Size option saves us the time and lowers the traffic.

If a client uses Transfer Size option with WRQ, it is the client who specifies in the option number of bytes to be transported. If the client uses the option with RRQ, it specifies value „0“ and a server changes this value to the corresponding one when responding with OACK.

Detailed description of the process in RFC 2349[3].

Chapter 3

Implementation

Both the client and server are written in the C++ programming language and are designed to run on a Linux-based operating system.

The whole implementation was written with object orienting programming paradigm in mind[1]. Because of that, most of the code is encapsulated in objects (classes mostly).

3.1 Common parts

There is a larger chunk of code that overlaps between client and server implementation and that is:

- TFTP packets,
- Options,
- I/O handlers,
- UDP communication,
- code to handle SIGINT,
- logging.

Each TFTP packet is a class that inherits from a generic class `TftpPacket` and implements two constructors: one that constructs the class from raw series of data and one where all the necessary data for the specific packet are given as arguments; and a method `MakeRaw()` which serializes the packet information into a stream of bytes as specified in TFTP protocol[5].

Options are parsed and stored in a class `Option`. Names of supported options are specified in an enumeration, but if an unsupported option appears, it is assigned enumeration value `UNSUPPORTED`. Because of this and a need to still remeber the option name for logging, its original string name is stored as well.

I/O handlers are split into two categories: file handlers and STD I/O handler. A generic class `FileHandler` contains methods such as checking if specified file already exists or a method to delete a file. Classes `Reader` and `Writer` inherit from it and extend it to do its respective work.

Two classes were created for UDP communication: `UdpClient` and `UdpServer`. `UdpClient` is used by TFTP client to transport its created packets across and is also used in the same

way by TFTP server after establishing connection with client. `UdpServer` is used by TFTP server to wait for incomming messages from outside.

Handling SIGINT proved to be challenging. Client is able to receive SIGINT with no bigger issue and when it comes, it handles it and signals its appearance using a global flag. The implementation often checks the flag and if it is true, potentially created and unfinished resources are cleaned and an error message is sent to server to signal an end to the communication. Server has it a little bit more complicated, as the SIGINT has to be signaled across all running communications with all clients, but the generic idea is the same.

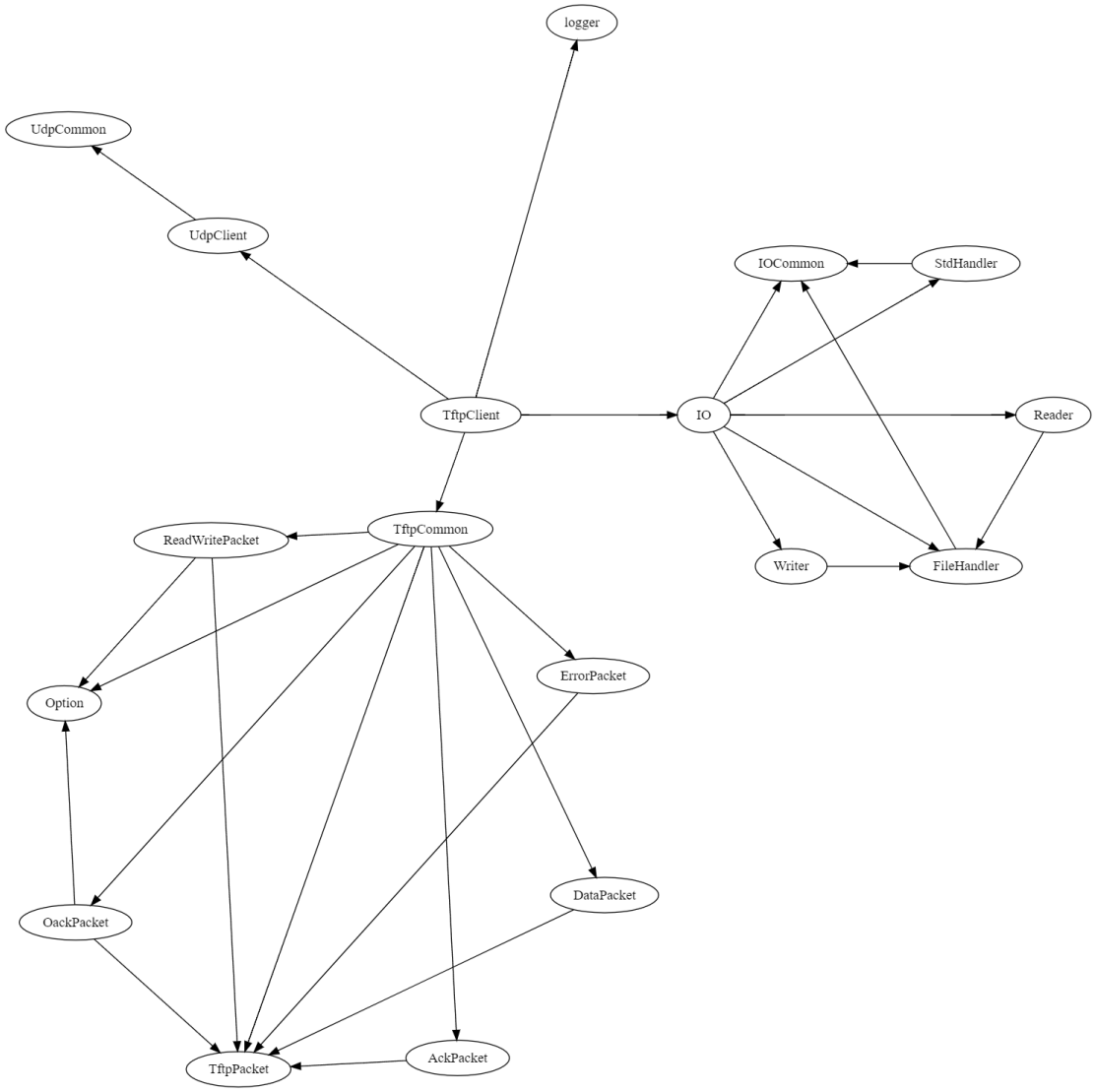
As the assignment requires, a logging of all incoming TFTP packets has to be implemented on client and server. The logs are printed on standard error output.

3.2 TFTP Client

Main method parses command line arguments and starts TFTP client, which then handles all the communication and I/O.

3.2.1 Graph Of Dependencies

This is a simplified graph which tries to showcase dependencies and/or usages between different classes and/or files of TFTP client.



3.2.2 How to Use

To use the TFTP client, follow the guidelines below:

```
tftp-client -h hostname [-p port] [-f filepath] -t dest_filepath
```

- **-h**: Specify the IP address or domain name of the remote server.
- **-p**: Define the port of the remote server. If not specified, the default port according to the specification will be assumed.

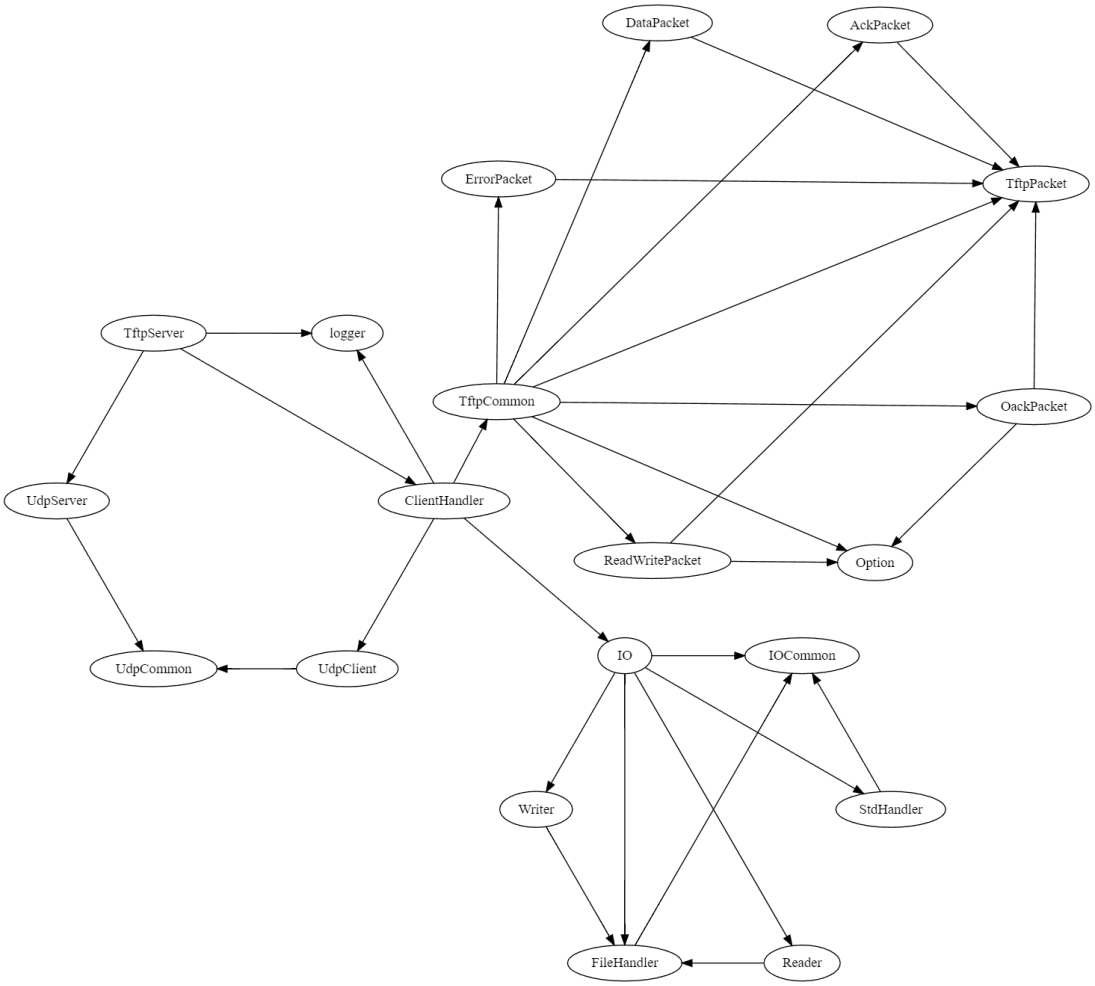
- **-f**: Specify the path to the file to be downloaded from the server (download). If not specified, the content from stdin will be used (upload).
- **-t**: Specify the path under which the file will be stored on the remote server or locally.

3.3 TFTP Server

Main method parses command line arguments and starts TFTP server, which is then stuck in infinite loop that waits for incoming packets and if any comes, it is given to a method which is run in a separate thread not to block the server itself. The packet is passed into `ClientHandler` class which handles the communication with the packet sender (client).

3.3.1 Graph Of Dependencies

This is a simplified graph which tries to showcase dependencies and/or usages between different classes and/or files of TFTP server.



3.3.2 How To Use

To use the TFTP server, follow the guidelines below:

```
tftp-server [-p port] root_dirpath
```

- **-p**: Specify the local port on which the server will expect incoming connections.
- **root_dirpath**: Specify the path to the root directory where incoming files will be stored.

3.4 Known limitations

As TFTP is a simple protocol, it and its implementation has its limits of what it can achieve:

- Maximum size of file to be sent - as TFTP uses DATA packets to transfer blocks of data and marks each DATA packet with incrementing block number, it is capable of transferring maximum of $2^{16} * 512 = 33554432$ bytes of data without options and with Block Size option maximum of $2^{16} * 65464 = 4290248704$ bytes of data. This is caused by the fact that block number is of size of two bytes.
- Might happen a loss of data - as TFTP uses UDP to transfer packets and UDP is not a reliable protocol, a loss of data might happen and even make the whole transfer impossible to finish.
- It is not capable of creating a directory hierarchy - no sources specify that if a destination path on server does not exist that server should create those directories, so if client tries to write into nonexistent directory, whole transfer fails.
- No possibility to rename or delete files - TFTP protocol does not give clients any way to rename or delete files on server.

Bibliography

[1] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1994.

[2] Palter, W. *TFTP Option Extension*. 2347. RFC Editor, 1998. Https://tools.ietf.org/html/rfc2347.

[3] Palter, W. *TFTP Timeout Interval and Transfer Size Options*. 2349. RFC Editor, 1998. Https://tools.ietf.org/html/rfc2349.

[4] Smith, K. and Clark, A. *TFTP Blocksize Option*. 2348. RFC Editor, 1998. Https://tools.ietf.org/html/rfc2348.

[5] Sollins, K. and Clark, D. *The TFTP Protocol (Revision 2)*. 1350. RFC Editor, 1992. Https://tools.ietf.org/html/rfc1350.