ISA – Project

TFTPv2 client

Adrián Kálazi
xkalaz00@stud.fit.vutbr.cz

November 6, 2021

Contents

Introduction

This document describes a TFTPv2 client implementation conforming to RFC1350 [1] with support for IPv4 as well as IPv6 connections.

This implementation contains the option extension [2] with support for the blocksize [3], timeout and transfer size [4] options.

The multicast [5] option as well as the obsolete mail mode are not supported.

1 Theory

The TFTP protocol (which stands for Trivial File Transfer Protocol) and its structure is specified in RFC1350 [1].

TFTP is implemented on top of the Internet User Datagram protocol and has no provisions for user authentication.

The communication begins with sending a communication initialization packet with a read (`RRQ`) or write (`WRQ`) request to the server. This packet contains the requested operation, filename and transmission mode. Transmission mode can be either `octet` for binary transfer without changing the sent data or `netascii` for the netascii format (which is ASCII [6] with modifications specified by the Telnet protocol [7])

In case of an error, the connection is terminated and an `ERROR` packet is sent.

The server can respond with a `DATA`, `ACK` or `ERROR` packet in case of a read request, write request or an error, respectively.

The client later responds with `ACK` or `DATA` packets, for read and write requests, respectively.

For consistency, each sent/received `DATA` block has its block id, which must be acknowledged by sending an `ACK` packet with the same block id.

The option extension defined in [2] extends the communication initialization packet by including options while retaining backwards compatibility with the base specification. These are specified as option/value pairs.

2 Implementation

The source code, written in C and located in `src`, is split into multiple source files each containing a logically separate block.

The client runs until an EOF is written to `stdin` or a fatal error occurs (such as an initial client allocation error).

Each line written to `stdin` is processed separately. The client handles transmission errors by informing the user and terminating the ongoing connection without affecting the client itself.

2.1 Input handling

Each transmission command is defined as string of options terminated by a newline.

The transmission command is processed as follows:

- Each whitespace is replaced by a zero byte, essentially making each word (option) a separate string
- The pointers to the beginning of each word are collected into a list of char pointers
- A counter is incremented for each word

This essentially results in an emulation of argc and argv which can then be supplied to getopt as ususal.

2.2 Option response handling

If the server chooses to ignore the `-t` (timeout) or `-s` (block size) options the timeout is simply ignored and the blocksize is reset to the default (512) as defined in [1]. The client also accepts block size offers by the server which are smaller than the value requested by the client, in which case the value offered by the server is used.

Some servers (such as `tftp-hpa`) which don't support the transfersize option on netascii transfers tend to generate an `ERROR` packet instead of ignoring the option. This behaviour is not specified in [4] and is considered a server bug. The client simply handles this as a normal transmission error by terminating the current transfer.

2.3 Format conversion

For netascii transfers the block size used by the client is considered as the encoded block size and the actual decoded block size may be equal or smaller depending on the content.

The total transfer size displayed on stdout with netascii transfers depends on the operation.

For `RRQ`, the displayed value is given by the server and represents the total netascii encoded size. The decoding is done on a per-message basis by replacing specific byte sequences with one byte replacements as defined in [7].

For `WRQ`, the displayed value is the size of the file before encoding as this doesn't impact the progress of the transfer in any way. The encoding is done sequentially for each byte from the given input file by replacing specific bytes with a byte sequence replacement as defined in [7], so the actual number of bytes read per block could be equal or less than the specified block size.

During both decoding and encoding, the message boundaries need to be considered in order to preserve data integrity.

3 Testing

After launch, the client accepts transfer commands which consist of transfer options terminated by a newline. Each transfer command opens a new connection to the server and the connection parameters are independent from other transfer commands.

The client terminates after receiving an `EOF`, which can be generated by `ctrl+D` on most terminal emulators.

The client was tested with two TFTP servers, namely `tftp-hpa`[1] and `rtftp`[2].

The data integrity was confirmed using `md5sum` for every combination of operations (`RRQ` / `WRQ`), transmission modes (`octet` / `netascii`) and internet protocol versions (`IPv4` / `IPv6`).

[1] `https://www.kernel.org/pub/software/network/tftp/tftp-hpa/`
[2] `https://github.com/reinerh/rtftp/`

References

[1] Dr. Karen R. Sollins. The TFTP Protocol (Revision 2). RFC 1350, July 1992.

[2] Gary S. Malkin and Art Harkin. TFTP Option Extension. RFC 2347, May 1998.

[3] Gary S. Malkin and Art Harkin. TFTP Blocksize Option. RFC 2348, May 1998.

[4] Gary S. Malkin and Art Harkin. TFTP Timeout Interval and Transfer Size Options. RFC 2349, May 1998.

[5] A. Thomas Emberson. TFTP Multicast Option. RFC 2090, February 1997.

[6] ASCII format for network interchange. RFC 20, October 1969.

[7] Telnet Protocol specification. RFC 764, June 1980.