Computer Networks

Laboratory project - Reliable Data Transmission over an Unreliable Network

Version 1.02 - 4.10.2018

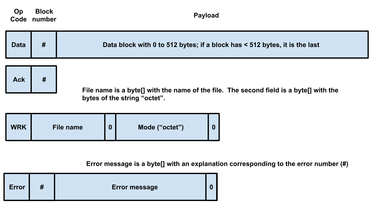
TFTP (Trivial File Transfer Protocol) is an IETF standardized protocol to transfer files from a client to a server and vice-versa. The goal of this project is to build a TFTP client program able to transfer any file, from a client machine, to a server machine where a standard TFTP server is running.

TFTP is an instance of the **stop & wait (S&W)** protocol for file transfer. That's why it is called a trivial protocol. It has a very simple specification, not many options, and is well known.

The Basic Trivial File Transfer Protocol

To help the reader, a brief introduction to TFTP follows. You can also consult the Wikipedia page on tftp. We will focus on the version of the protocol without options, using data blocks of at most 512 bytes, see RFC 1350 (search for this RFC with your preferred search engine).

TFTP messages, often called TFTP Packets, are encapsulated in UDP Datagrams. They have an header and a payload. The following figure shows them.



The two most important messages to support the **S&W** protocol are the **DATA** and **ACK** messages, both with the following header:

- Operation Code - 2 bytes - a short
- Sequence Number - 2 bytes - a short - a cumulative sequence number starting at 1

The Data message has a payload containing a byte array of at most 512 bytes according to RFC 1350 (or grater in more recent versions). A zero byte payload is also valid. A data message with a payload of size < 512 bytes contains the last bytes of the file. An empty payload is also valid.

"Any transfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes" (in RFC 1350).

To start the transfer, the client must send an initial message to the server with the file name to transfer, as well as a transfer mode. This message has an Operation Code (2 bytes), a byte array with the character codes of a string with the file name, ended with a zero valued byte.

The name of the file is followed by a transfer mode, also a byte array with the character codes of the transfer mode, ended with a zero valued byte. We will only use the mode "octet", where the file is transferred as is, i.e., the origin and destination files are fully identical.

Before the transfer from the client to the server starts, this initial message must be acknowledged by the server with an ACK message with a 0 sequence number.

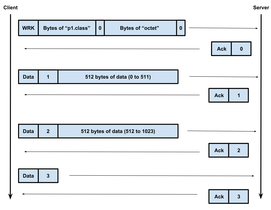
The figure also shows a special message, the error message, sent when something went wrong. Operation codes values and error message values are shown below.

Operation codes

Name	Meaning	Value
RRQ	Read Request	1 (the client requests a file from the server)
WRQ	Write Request	2 (the client wants to send a file to the server)
DATA	Data Message	3
ACK	Acknowl. Message	4
ERROR	Error Message	5

Value	Meaning
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 (... port)
6	File already exists.
7	No such user.

The following figure presents a diagram of an upload of a file of size 1024 bytes (when no messages are lost).



Mandatory delivery of your project

You must deliver a program (**TFtp.java**) executed the following way: `java TFtp <server> <port> <filename>`

It must be able to transfer the file `<filename>` to a standard TFTP server running at the host with IP address `<server>` (e.g. localhost), and accepting requests at port number `<port>`.

Note 1 - the standard TFTP server accepts requests at port 69.

Note 2 - your program must follow the tftp RFC and be aware that the server will use **a different port** for each transfer, and will send datagrams to the client using that source port.

If the transfer ends with success, the client will output the following data:

- Number of transferred bytes (size of the file)
- Total duration of the file transfer (in ms) and average end to end transfer rate (in bps)
- Number of data messages sent (including repetitions)
- Number of all ack messages received (including the initial one)

Optional deliveries of your project

Block size option - to speedup the transfer rate, your sender is able to use a data block larger than 512 bytes. It has an extra last parameter with that size:
java MyTftp host port filename blksize

In fact, TFTP versions developed after RFC 1350, the initial message exchange allows the client and server to negotiate several options, namely a different maximum block size, i.e., using data blocks larger than 512 bytes. See RFC 2348.

Dynamic timeout value option - to speedup error recovery, your sender uses a dynamically estimated timeout value.

GBN option - your sender uses the GBN protocol.

A standard **S&W** receiver is also able to receive a file sent by a sender using a **GBN sliding window protocol**. Therefore, using a standard tftp receiver, a sender of a file may use a pure **S&W** protocol or, even if the receiver only has one slot available to the next data block, it can also use a sliding window protocol of the go-back-N (**GBN**) version to recover from lost packets.

With these options, your program has some or all of the following extra output:

- Min, average and max RTT between the client and the server (in ms)
- Min, average and max timeout values used by the client (in ms)
- Min, average and max size of the client window (in blocks)

Options are listed in their increasing implementation difficulty. We suggest that you start with a solution that does not implement any options.

Grading

- A program that does not transfer files or transfers files **incorrectly**, will be graded at most 8 (0 - 20 scale);
- A program that transfers files **correctly**, with no extra implementation options, will be graded at most 14;
- Code clarity and structure is also accounted for grading purposes.

Projects may be developed in groups of up to two students. However, they must be delivered **individually** and will be graded **individually**. Final grading of the delivered project will be averaged with the results of the project written test.

Project delivery

Please, recall the rules concerning this project

- The project can be completed by a student or by a group of up to two students. However, each student **must** deliver his own copy of his/her (or the group) results and files. The submission form allows the identification of the other member of the group if he/she exists.
- Submissions of the project results are allowed from October 14 (Sunday) up to midnight of October 16 (Tuesday). Late delivery, with penalisation of 1 grade per day, is allowed upto October 19 (Friday) by the end of the day.
- Students may have a copy of their sources to consult during the project evaluation mini-test that. This test will take place after the theoretical test in the morning of October 20th.

You will find the project delivery form as well as an example of a dummy submission here

Additional materials

The following extra materials are available.

- A java class to assemble and decode TFTP packets: TFtpPacketV18.
- A template of the class Stats that you must use to print your statistics after client execution.
- Three files to test your client:
  - file2048 a file with 2048 bytes
  - file16384 a file with 16384 bytes
  - file1000000 a file with 1000000 bytes

Bibliography

RFCs 1350 and 2348 - http://www.ietf.org/
Wikipedia page on tftp - https://en.wikipedia.org/wiki/Trivial_File_Transfer_Protocol

Hints

Start by understanding how the class TFtpPacketV18 may be used to assemble and decode TFTP packets. Build a first sketch of your solution that only sends a file with less then 512 bytes. Learn how to deal with timeouts and retransmissions. Plan and build your solution without options. If it is stable and correct, try to implement some of or all the options.

Next, you will find some code excerpts to help you start the project.

Main Method

Below, there is an example of the main method of a TFtp class that parses the command line arguments.

In [ ]:
```
public static void main(String[] args) throws Exception {
    int port;
    String fileName;
    InetAddress server;

    switch (args.length) {
        case 3:
            server = InetAddress.getByName(args[0]);
            port = Integer.valueOf(args[1]);
            fileName = args[2];
            break;
        default:
            System.out.printf("usage: java %s server port filename\n", TFtp.class.getName());
            return;
    }

    Stats stats = new Stats();
    sendFile(stats, fileName, server, port);
    stats.printReport();
}
```

Testing

You can test your solution against a server running in a Docker container, listening on port **6969**. For more information on Docker, check this page.

To use the prepared image, install docker in your machine. Once installed, execute the following commands in terminal/console.

To update the image to the latest version:

```
docker pull smduarte/rc18-trab1-tftp-test1
```

- Linux (A)

```
docker run --privileged -p 6969:69/udp -ti smduarte/rc18-trab1-tftp-test1
```

- Windows (Docker Toolbox)

```
winpty docker run --privileged -p 6969:69/udp -ti smduarte/rc18-trab1-tftp-test1
```

- MacOS / Windows (Native Docker) / Linux (B)

```
docker run --privileged -p 6969:6969/udp -ti smduarte/rc18-trab1-tftp-test1
```

The IP address of the server is 127.0.0.1 in MacOS and Windows Native, Linux(B), while the port is 6969. In Windows Toolbox, you have to use the IP address shown when Docker is launched (usually, 192.168.99.100).

In Linux (A), the IP address is 172.XXX.YYY.2. Use the command `ip addr` to find out the XXX, YYY. **The port is 69**.

Note. In some versions of Windows and in some machines, you may have to install the Toolbox version of Docker. Moreover, in some cases, CPU virtualization support is not enabled by default and needs to be enabled in the BIOS/EFI.

Verify the transfer

Use the following steps to confirm the transfer completed and the contents of destination file matches original file.

1. Compute, once, the *md5 checksum* of each of the files being transferred. You can do it online or you can install a MD5 utility for your platform.
2. Execute `docker ps` to obtain the `<CONTAINER ID>` of the TFTP Server.
3. Execute `docker exec -t <CONTAINER ID> md5sum /tmp/<filename>`
   - If the transfer was successful, the two *md5 checksums* should be the same.
4. To list the files (and their lengths) already stored by the TFTP server, use:

   ```
   docker exec -t <CONTAINER ID> ls -al /tmp/
   ```