De La Salle University - Manila
Second Trimester A.Y. 2019-2020

# Trivial File Transfer Protocol (TFTP) Client Program and Documentation

Submitted by:

**Javier, Maria Patricia G.**

**NSCOM01 - S12**

Submitted on:

**April 25, 2020**

Submitted to:

**Ms. Arlyn Verina Ong**

# NSCOM01 PROJECT DOCUMENTATION

## Introduction

File Transfers have been used as a common feature of many applications nowadays. There are two known protocols for transferring files between two systems or hosts namely: File Transfer Protocol also known as FTP and Trivial File Transfer Protocol also known as TFTP. These two have the same functionality as a protocol used for information and file transfers from a client to a server or vice versa but differs on data transfer reliability as FTP uses TCP for establishing connection while TFTP uses UDP as their transport layer protocol.

Trivial File Transfer Protocol (TFTP) does file transfer without user authentication and uses UDP port 69 by default. TFTP specifications include mechanisms that provide reliable and ordered data delivery with simplified requirements to applications that would have otherwise been unavailable from the transport layer. The majority of TFTP applications conform to TFTP version 2 which has its specifications documented in RFC 1350 and later extended with additional features using RFCs 2347, 2348 and 2349.

## Objectives

The objective of the project was to implement a TFTP Client program that complies with the protocol specifications documented in RFC 1350. With a user interface, a client must be able to run the program and make use of the following features:

1) File Upload from client to server
2) File Download from server to client
3) Proper Error Handling
4) Option to customize transfer block size (Optional)

## Design Methodology

In compliance with the protocol specifications documented in RFC 1350, this project implemented a TFTP Client program and was coded using Python. Designing of the software adapted the packet formats of the RFC 1350 TFTP version 2. Packet creation and logging of the transfers was coded and can be viewed in TFTPPacket.py source code wherein packets are to be formatted, sent, and received as:

```
TFTP Packets

   TFTP supports five types of packets:

          opcode  operation
            1     Read request (RRQ)
            2     Write request (WRQ)
            3     Data (DATA)
            4     Acknowledgment (ACK)
            5     Error (ERROR)

   The TFTP header of a packet contains the opcode associated with that
packet.
```

*Figure 1: RFC 1350 Page 5 - Packet Opcodes*

```
   Type   Op #     Format without header

          2 bytes    string   1 byte     string   1 byte
          -----------------------------------------------
   RRQ/  | 01/02 |  Filename  |   0  |    Mode    |   0  |
   WRQ    -----------------------------------------------
          2 bytes    2 bytes       n bytes
          ---------------------------------
   DATA  | 03    |   Block #  |    Data    |
          ---------------------------------
          2 bytes    2 bytes
          -------------------
   ACK   | 04    |   Block #  |
          --------------------
          2 bytes  2 bytes        string    1 byte
          ----------------------------------------
   ERROR | 05    |  ErrorCode |   ErrMsg   |   0  |
          ----------------------------------------
```

*Figure 2: RFC 1350 Page 9 – Different Packet Formats*

In addition to the program features, option to customize the transfer block size was implemented with compliance to the specifications documented in RFCs 2347 and 2348 wherein additional packet formats were considered, namely: OACK Packet and Block Size Extensions.

```
      +-------+---~~---+---+---~~---+---+---~~---+---+---~~---+---+
      |  opc  |  opt1  | 0 | value1 | 0 |  optN  | 0 | valueN | 0 |
      +-------+---~~---+---+---~~---+---+---~~---+---+---~~---+---+

      opc
         The opcode field contains a 6, for Option Acknowledgment.

      opt1
         The first option acknowledgment, copied from the original
         request.

      value1
         The acknowledged value associated with the first option.  If
         and how this value may differ from the original request is
         detailed in the specification for the option.

      optN, valueN
         The final option/value acknowledgment pair.
```

*Figure 3: RFC 2347 Page 3 – OACK Packet Format*

```
Blocksize Option Specification

   The TFTP Read Request or Write Request packet is modified to include
   the blocksize option as follows.  Note that all fields except "opc"
   are NULL-terminated.

      +-------+---~~---+---+---~~---+---+---~~---+---+---~~---+---+
      |  opc  |filename| 0 |  mode  | 0 | blksize| 0 | #octets| 0 |
      +-------+---~~---+---+---~~---+---+---~~---+---+---~~---+---+

      opc
         The opcode field contains either a 1, for Read Requests, or 2,
         for Write Requests, as defined in [1].
      filename
         The name of the file to be read or written, as defined in [1].

      mode
         The mode of the file transfer: "netascii", "octet", or "mail",
         as defined in [1].

      blksize
         The Blocksize option, "blksize" (case in-sensitive).

      #octets
         The number of octets in a block, specified in ASCII.  Valid
         values range between "8" and "65464" octets, inclusive.  The
         blocksize refers to the number of data octets; it does not
         include the four octets of TFTP header.
```

*Figure 4: RFC 2348 Page 1-2 – RRQ/WRQ Packet with Block Size Option Extension*

If block size was customized into a size other than 512, a RRQ/WRQ Packet extended with a Block Size Option will be sent to the server and server shall respond with an OACK Packet, acknowledging the block size requested and proceeds to the transfer of DATA and ACK packets to be exchanged between the client and the server for upload or download.

```
Read Request

      client                                           server
      -------------------------------------------------------
      |1|foofile|0|octet|0|blksize|0|1432|0|  -->               RRQ
                                    <--  |6|blksize|0|1432|0|   OACK
      |4|0|  -->                                                ACK
                             <--  |3|1| 1432 octets of data |   DATA
      |4|1|  -->                                                ACK
                             <--  |3|2| 1432 octets of data |   DATA
      |4|2|  -->                                                ACK
                             <--  |3|3|<1432 octets of data |   DATA
      |4|3|  -->                                                ACK


Write Request

      client                                           server
      -------------------------------------------------------
      |2|barfile|0|octet|0|blksize|0|2048|0|  -->               WRQ
                                    <--  |6|blksize|0|2048|0|   OACK
      |3|1| 2048 octets of data |  -->                          DATA
                                                   <--  |4|1|   ACK
      |3|2| 2048 octets of data |  -->                          DATA
                                                   <--  |4|2|   ACK
      |3|3|<2048 octets of data |  -->                          DATA
                                                   <--  |4|3|   ACK
```

*Figure 5: RFC 2347 Page 5 – Example of Read and Write Requests with Block Size Option Extension*

## Program Implementation

The program can be used when RunMe.py is ran by the user. The program, when ran, provides a graphical user interface wherein fields for specifying the server address, transfer block size, remote filenames for upload and download, and preferred filenames that are applicable to both upload and download.

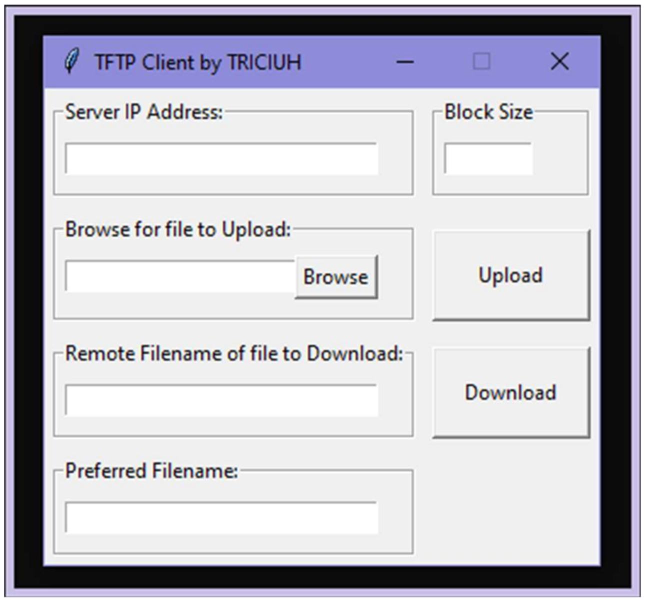

*Figure 1: Graphical User Interface*

GUI Features:

1) Server IP Address is required and shall be filled up by the user in order to use the features upload and download.
2) Block Size is an optional field that sets the transfer block size to 512 if not filled up by the user. If filled up, then the program shall use the input block size for the file transfer.
3) Upload feature provides 'Browse' button for easier navigation of file to upload. Manual typing of file to upload is also acceptable. Upload button for starting upload.
4) Download feature requires a remote filename to be provided by the user for successful download of file/s. Download button for starting download.
5) Preferred filename is an optional field that sets the local filename when file is to be saved on download and sets the preferred filename of the file to be saved as in the server on upload. If not filled up, the remote file name will be set as the preferred filename.

Upon using the program for file transfers, both upload and download, messages informing the client of the process being done are shown on the command line interface provided with what action was done, what are the parameters involved, date and time of the file transfer, and a message to inform the user of the program's status. This then will be logged and will provide a text file named TFTPLog.txt for reference purposes of the transactions made using the program.

## I. Message Format Details

When Upload or Download button was clicked by the user, request packet will be created with the corresponding opcode according to which button was pressed as shown below.

```
Download (Creates RRQ Request Packet)
           --------------------------------------------
          | 01 |  Filename  |   0  |    Mode    |   0  |
           --------------------------------------------
Upload  (Creates WRQ Request Packet)
           --------------------------------------------
          | 02 |  Filename  |   0  |    Mode    |   0  |
           --------------------------------------------
```

*Figure 1.1: Message Format to be sent as request packet when block size is 512*

```
Download (Creates RRQ Request Packet with Block Size Option)
     ----------~~-----------~~-----------~~-----------~~-------
    |  01  |filename| 0 |  mode  | 0 | blksize| 0 | #octets| 0 |
     ----------~~-----------~~-----------~~-----------~~-------
Upload  (Creates WRQ Request Packet with Block Size Option)
     ----------~~-----------~~-----------~~-----------~~-------
    |  02  |filename| 0 |  mode  | 0 | blksize| 0 | #octets| 0 |
     ----------~~-----------~~-----------~~-----------~~-------
```

*Figure 1.2: Message Format to be sent as request packet when block size is not 512*

```
Logged action: <UPLOAD/DOWNLOAD>
Date: <YYYY-MM-DD> <Exact Time>
Params: (<Block Size>, <Remote Filename>, <Preferred Filename>, 'octet')
Message: Initiating WRQ request (<Server IP Address>, 69)/<filename> of
size <size in KB> KB.
```

*Figure 1.3: Message in command line to indicate sending of request to server*

Upon receiving the request packet, if the request was to write with block size set to 512, the server will then respond with an ACK packet of block number 0 to indicate the start of the file transfer. The ACK packet is formatted as shown below.

```
Acknowledgement (Initialized at block # 0 to start sending DATA packet)
                     --------------------
                    |  04  |   Block #   |
                     --------------------
```

*Figure 1.4: Message Format to be sent as ack packet when block size is 512*

On the other hand, if the request was to read with block size set to 512, the server will proceed to sending the DATA packet and to be responded by the client with an ACK packet to the corresponding block number of data sent by the server. The DATA packet is formatted as shown below.

```
Data
                   ----------------------------------
                  | 03    |   Block #  |    Data     |
                   ----------------------------------
```

*Figure 1.5: Message Format to be sent as data packet*

As for file transfers that have block size set to a value that is not 512, a RRQ/WRQ packet with a Block Size Option will be sent by the client as a request packet to the server. The RRQ/WRQ packet with a Block Size Option is formatted as shown in Figure 1.2 which will be responded by the server with an OACK packet that acknowledges the requested block size. The OACK Packet is formatted as shown below.

```
Option Acknowledgement
             --------------------------------------------
            |  06  |  blksize  | 0 | blksize value | 0 |
             --------------------------------------------
```

*Figure 1.6: Message Format to be sent as oack packet*

After an OACK packet was received by the client, then the transfer proceeds to normal exchange of DATA and ACK packet but with a different block size set to be received as size of data being sent.

If errors on exchange of packets within the transaction occurs, the transfer will terminate itself by sending out an ERROR packet to the client indicating that the transfer shall now be terminated because an error was encountered during the exchange of packets. The ERROR packet shall be formatted as shown below.

```
Error
                                ----------------------------------------
                              | 05    |  ErrorCode |   ErrMsg   |   0  |
                                ----------------------------------------
```

*Figure 1.7: Message Format to be sent as error packet*

## A. Successful Connection Established to the Server and Successful File Transfer

### a. Upload

If successful connection was established with the server, the program automatically proceeds with the client sending the DATA packet to the server and receiving ACK packet from the server. Through this, file will be written to the server for upload and packets will be exchanged continuously. The next message that will inform the user of the file transfer progress is when the last packet was reached.

```
Logged action: Last Packet Reached
Date: <YYYY-MM-DD> <Exact Time>
Params: None
Message: Terminating write...
```

*Figure A.a.1: Message in command line to indicate reaching last packet of data for upload*

By reaching the last packet, it means that the writing of file to the server was successful until the last chunk of data and that the upload was a success. The user will then be informed of a successful upload.

```
Logged action: UPLOAD SUCCESS:
Date: <YYYY-MM-DD> <Exact Time>
Params: (<Block Size>, <Remote Filename>, <Preferred Filename>,
'octet')
Message: <Filename> to host (<Server IP Address>, 69),
total bytes sent: <total bytes>,
total retry counts: <# of transfer retry because of error>,
execution time: <total seconds of transaction> seconds
```

*Figure A.a.1: Message in command line to indicate a successful upload*

This will then lead to the closing of socket and connection termination with the server.

**b. Download**

If successful connection was established with the server, the program automatically proceeds with receiving DATA packets from the server and the client will response with an ACK packet for each data received with the corresponding block number to acknowledge. If no errors were encountered during the file transfer and the last packet was reached, it means that the reading of file from the server was successful until the last chunk of data and that the download was a success. The user will then be informed of a successful upload.

```
Logged action: DOWNLOAD SUCCESS:
Date: <YYYY-MM-DD> <Exact Time>
Params: (<Block Size>, <Remote Filename>, <Preferred Filename>,
'octet')
Message: <Filename> to host (<Server IP Address>, 69),
total bytes sent: <total bytes>,
total retry counts: <# of transfer retry because of error>,
execution time: <total seconds of transaction> seconds
```

*Figure A.b.1: Message in command line to indicate a successful download*

This will then lead to the closing of socket and connection termination with the server.

## B. Unsuccessful Connection to the Server and Unsuccessful File Transfer

### a. No Active Server / Failed Connection to Server IP Address

If establishing of connection was unsuccessful because the provided Server IP Address is not an existing server or could not be connected to, a message will inform the user that the program could not connect to the server.

```
Logged action: <DOWNLOAD/UPLOAD> FAILED: Could not connect
to the server.
Date: <YYYY-MM-DD> <Exact Time>
Params: (<Block Size>, <Remote Filename>, <Preferred
Filename>, 'octet')
Message: Error timed out
```

*Figure B.a.1: Message in command line to indicate a failed upload or download*

This will then lead to the closing of socket and connection termination with the server.

### b. File Not Found

If establishing of connection was unsuccessful because the provided Remote Filename is not an existing file on the client's or server's directory, a message will inform the user that the file was not found.

```
Logged action: <DOWNLOAD/UPLOAD>
Date: <YYYY-MM-DD> <Exact Time>
Params: (<Block Size>, <Remote Filename>, <Preferred
Filename>, 'octet')
Message: File Not Found.
```

*Figure B.b.1: Message in command line to indicate a failed upload or download*

This will then lead to the closing of socket and connection termination with the server.

### C. Terminating Connection to the Server and Socket Closing

To inform the user of a connection termination and closing of socket, a message will be shown in the command line.

```
Logged action: Ending Connection…
Date: <YYYY-MM-DD> <Exact Time>
Params: None
Message: Connection Closed.
```

*Figure C.1: Message in command line to indicate a failed upload or download*

## II. Message Sequence Diagrams

### a. Upload

#### i. Default 512 Transfer

```
UPLOAD with 512 as block size (default)

      client                                           server
      --------------------------------------------------------
      |2|filename|0|octet|0| -->                                  WRQ
                                                   <--  |4|0|    ACK
      |3|1| 512  octets of data |  -->                            DATA
                                                   <--  |4|1|    ACK
      |3|2| 512  octets of data |  -->                            DATA
                                                   <--  |4|2|    ACK
      |3|3| 512  octets of data |  -->                            DATA
                                                   <--  |4|3|    ACK
                                  . . .

      |3|n| <512 octets of data (last) |  -->                     DATA
                                                   <--  |4|n|    ACK
```

*Figure a.i: Sequence Diagram of an upload with 512 as block size (default)*

### ii. Optional Block Transfer (block size != 512)

```
UPLOAD with 600 as block size (default)

      client                                           server
      -------------------------------------------------------
      |2|filename|0|octet|0|blksize|0|600|0|  -->               WRQ
                                     <--  |6|blksize|0|600|0|   OACK
      |3|1| 600  octets of data |  -->                          DATA
                                                   <--  |4|1|   ACK
      |3|2| 600  octets of data |  -->                          DATA
                                                   <--  |4|2|   ACK
      |3|3| 600  octets of data |  -->                          DATA
                                                   <--  |4|3|   ACK
                                   . . .

      |3|n| <600 octets of data (last) |  -->                   DATA
                                                   <--  |4|n|   ACK
```

*Figure a.ii: Sequence Diagram of an upload with 600 as block size*

## b. Download

### i. Default 512 Transfer

```
DOWNLOAD with 512 as block size (default)

      client                                           server
      -------------------------------------------------------
      |1|filename|0|octet|0| -->                                RRQ
                             <--  |3|1| 512  octets of data |   DATA
      |4|1|  -->                                                ACK
                             <--  |3|2| 512  octets of data |   DATA
      |4|2|  -->                                                ACK
                             <--  |3|3| 512  octets of data |   DATA
      |4|3|  -->                                                ACK
                                   . . .

                             <--  |3|n| <512 octets of data |   DATA
      |4|n|  -->                                                ACK
```

*Figure b.i: Sequence Diagram of a download with 512 as block size (default)*

### ii. Optional Block Transfer (block size != 512)

```
DOWNLOAD with 600 as block size

      client                                               server
      -------------------------------------------------------
      |1|filename|0|octet|0|blksize|0|600|0|  -->              RRQ
                                  <--  |6|blksize|0|600|0|    OACK
      |4|0|  -->                                               ACK
                           <--  |3|1| 600  octets of data |    DATA
      |4|1|  -->                                               ACK
                           <--  |3|2| 600  octets of data |    DATA
      |4|2|  -->                                               ACK
                           <--  |3|3| 600  octets of data |    DATA
      |4|3|  -->                                               ACK
                                 . . .

                           <--  |3|n| <600 octets of data |    DATA
      |4|n|  -->                                               ACK
```

*Figure b.ii: Sequence Diagram of a download with 600 as block size*

### c. Errors

#### i. No Server

If no server can be found or connected to, there will be no recipient for the RRQ/WRQ Request Packet. Therefore, there will be no packets to be exchanged between client and server to begin with.

#### ii. File Not Found

##### 1. Upload

```
UPLOAD of a nonexistent file

      client                                               server
      -------------------------------------------------------
      |2|filename|0|octet|0| -->                               WRQ
                     <--  |5|File not found|File not found|0|   ERROR
```

*Figure c.ii.1: Sequence Diagram of an upload of a nonexistent file*

## 2. Download

```
DOWNLOAD of a nonexistent file

      client                                            server
      -------------------------------------------------------
      |1|filename|0|octet|0| -->                                RRQ
                     <--  |5|File not found|File not found|0|   ERROR
```

*Figure c.ii.2: Sequence Diagram of a download of a nonexistent file*