**Design of tftpd**

**Revision History**

| Revision | Authors | Contact Info | Description | Date |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

[1. Introduction 2](#_Toc17722)

[1.1 Purpose 2](#_Toc27484)

[1.2 References 2](#_Toc30055)

[1.3 Terminology 2](#_Toc26084)

[2. Description and Specifications of Functionalities 3](#_Toc7361)

[2.1 Background 3](#_Toc675)

[2.2 Configuration and Management Commands 3](#_Toc1855)

[2.2.1 Enable TFTP Server 3](#_Toc5522)

[2.2.2 Show the Configurations and Status 3](#_Toc6483)

[2.3 Functional Requirements 3](#_Toc14416)

[2.3.1 Requirement (behavior) 1 3](#_Toc31051)

[2.3.2 Requirement (behavior) 2 3](#_Toc18037)

[2.3.3 Requirement (behavior) 3 3](#_Toc12502)

[2.4 Performance Requirements 3](#_Toc13701)

[3. Architecture 4](#_Toc28595)

[3.1 Component 1 4](#_Toc28004)

[3.2 Component 2 4](#_Toc8401)

[4. Data 5](#_Toc11641)

[4.1 Macro Definitions 5](#_Toc19814)

[4.1.1 Message Types 5](#_Toc11236)

[4.1.2 Packet Types 5](#_Toc26055)

[4.2 Data Structures 5](#_Toc18295)

[4.2.1 TFTP Session (example) 5](#_Toc1548)

[4.3 Main Global Variables 5](#_Toc10828)

[4.3.1 TFTP Session Variable (example) 5](#_Toc15556)

[5. Procedures 6](#_Toc27178)

[5.1 Main Process of the Main Task (example) 6](#_Toc19517)

[5.2 Create a New Session (example) 6](#_Toc23846)

[6. Interfaces 7](#_Toc25397)

[6.1 abc 7](#_Toc19479)

# Introduction

## Purpose

TFTP is a very simple protocol used to transfer files. It is from this that its name comes, Trivial File Transfer Protocol or TFTP. Each non-terminal packet is acknowledged separately. This document describes the protocol and its types of packets. The document also explains the reasons behind some of the design decisions.

TFTP is a simple protocol to transfer files, and therefore was named the Trivial File Transfer Protocol or TFTP. It has been implemented on top of the Internet User Datagram protocol (UDP or Datagram) [2] so it may be used to move files between machines on different networks implementing UDP. (This should not exclude the possibility of implementing TFTP on top of other datagram protocols.) It is designed to be small and easy to implement. Therefore, it lacks most of the features of a regular FTP. The only thing it can do is read and write files (or mail) from/to a remote server. It cannot list directories, and currently has no provisions for user authentication. In common with other Internet protocols, it passes 8 bit bytes of data. Three modes of transfer are currently supported: netascii (This is ascii as defined in "USA Standard Code for Information Interchange" [1] with the modifications specified in "Telnet Protocol Specification" [3].) Note that it is 8 bit ascii. The term "netascii" will be used throughout this document to mean this particular version of ascii.); octet (This replaces the "binary" mode of previous versions of this document.) raw 8 bit bytes; mail, netascii characters sent to a user rather than a file. (The mail mode is obsolete and should not be implemented or used.) Additional modes can be defined by pairs of cooperating hosts. Reference [4] (section 4.2) should be consulted for further valuable directives and suggestions on TFTP.

(Referring to the following text, you can modify it according to your own needs.)

The section "Description and Specifications of Functionalities" in Chapter 2 describes the background and operational behaviors of the module tftpd, which is the input for design and implementation, and also the basis for testing and technical support.

Chapter 3-6 specifies the design of tfpd.

## References

1. RFC1350, 2. RFC2347, 3. RFC2348, 4. RFC2349, 5. RFC23

## Terminology

TFTP: Trivial File Transfer Protocol.

WRQ: Write request

RRQ: read request.

ACK: acknowledgement.

# Description and Specifications of Functionalities

## Background

## Configuration and Management Commands

### Enable TFTP Server

Command: [no] tftp server enable

Configuration mode: Global configuration mode

Description: The function enables tftp server for the switch. We can use “no” infornt of the command to disable it.

Parameters: no parameter needed.

Default: none

### Show the Configurations and Status

Command: show tftp server

Configuration mode: Privileged mode and global configuration mode

Description: Display all the current configuration of the tftp server such as port no, timeout value, retry count, block and window size.

Parameters: none

Default: none

### Set UDP port of TFTP server

Command: [no] tftp server port {port}

Configuration mode: Privileged mode and global configuration mode

Description: Change the value of the default port of the tftp server. “no” will set the port back to default value: none

Parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| port | port number for tftp request. an int value. ranged from 0 to 65,535. |

Default: default port number is 69 in decimal.

### Configure timeout and retry count for re-transmission

Command: [no] tftp server retransmit {timeout} {retry}

Configuration mode: Privileged mode and global configuration mode

Description: set the value of timeout and number of retry. “no” for seting the value back to default.

Parameters:

|  |  |
| --- | --- |
| Parameter | Description |
| timeout | time (seconds) to wait for a replay. range 1~255 inclusive. |
| retry | number of tries to communicate with with client if connection fails. range 1~6 inclusive. |

Note: timeout × retry <= 255;

Default: timeout : 3 second, retry: 3 tries.

### Saving configurations:

Command: write

Configuration mode: Privileged mode and global configuration mode

Description: save the configureation of the tftp module to the config file.

Parameters: none

Default: none

### Show running configurations:

Command: show running-config

Configuration mode: Privileged mode and global configuration mode

Description: show the runing config file with configurable tftp information including port number, timeout and retry value.

Parameters: none

Default: none

Command: show version all, show version module tftpd

Configuration mode: Privileged mode and global configuration mode

Description: Check the current tftp module version.

Parameters: none.

Default: none.

## Functional Requirements

### Requirement (behavior) 1

An independent module which will work as a TFTP server. The server will use TFTP protocol to upload and download files to and from a switch.

### Requirement (behavior) 2

The server will be able to serve 3 clients at the same time for the read request of same file. But a read and write session on a same file should not occur simultaneously. And two or more write session in a file at the same time is also invalid.

### Requirement (behavior) 3

Server in the switch can be enabled and disabled by command.

### Requirement (behavior) 4

Support configurable UDP port of tftp server. configuration is done using command.

### Requirement (behavior) 5

Support configurable timeout and retry count for re-transmission. The default values of the two parameters are 3 seconds and 3 times respectively. The value range of timeout is 1~255 inclusive, and retry 1~6 inclusive, and their product cannot be greater than 255. Note that the retry value includes the first normal transmission.

### Requirement (behavior) 6

Support negotiation of block-size. According to RFC2348. The default block-size is 512 bytes. But it can be improved if client wants. According to the RFC2348 we can use block-size of 1k to 8k for improving the transfer rate. But it should always be less or equal to the requested rate.

### Requirement (behavior) 7

### Support negotiation of window size. From RFC7440 we can get the window size which can be between 1 to 65535. If there is no block size specified it will be the default value of 512 bytes. The RFC7440 shows that increasing window size for a certain limit can improve performance. after that limit the performance improvement is negligible. So according to RFC7440 the window size of 32 is suitable for block-size of 1456.

## Performance Requirements

For files of the same size, in a single session scenario, the upload and download time is not more than the time of the existing TFTP client method, i.e. the method through the command “copy tftp”. The “copy tftp” command transfer file of 5402131 byte in 18 seconds. so the transfer rate should be 300119 bytes/sec or 301kbps (approx).

For handling three read request handling, the download time of one session is not more than 125% of the download time for a file of the same size in a single session scenario. So it will be 375 Kbps (approx).

# Architecture

The module is for tftp sever implementation. It will be like a application software so its position will be along with other basic application running in he software system for this the priority of the main task of the module will be 128 or average priority. It has relationship with the system module as it depends on it for some commands such as “show running-config”, “write”, “show version all” and so on. It also depends on the file operation routines. The module has some basic componets the are described as:

## Component 1: Request Receiver

The request receiver is the main task which will stay open all the time and listen to the port specified for our tftp module. By default it will accept any request at port 69. Then it will validate the request and send error reply if the packet is invalid. If valid request is received it will forward it to the Connection establisher.

## Component 2: Connection Establisher

After receiving the request from the previous component it will create a new task and next operations will happen in the task. The task will create a socket with random port number. This socket will be used for further communication. Then check for options in the received request. If there are any valid options then OACK reply will be sent. If no valid option is found then The request will be forwarded for file processing.

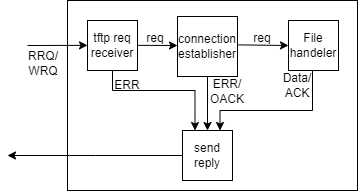


Fig 3.1: Relationship and dependency of each components.

## Component 3: File Handler

File processor checks for the opcode type. if it’s a RRQ then it will check if the file is valid to open. If valid then it will collect the data and send it as data block and receive ACK after each block. Else it will send error packet. If its WRQ request then it will create a file and send ACK with block number 0. Then accept data block and store data in it. After receiving each valid block it sends ACK to the client. During writing all other operations are halt.

## Component 4: Send Reply

This component will gather reply form the other components and order them properly and send them to the client.

# Data

## Macro Definitions

### Message Types

### Packet Types

#define ...…

### Max value

#define MAX\_BLKSIZE 1468

#define MAX\_WINDOWSIZE 32

#define FILE\_NAME\_LENGTH 32

#define BUFFER\_SIZE 1500

### Others

#define PRIORITY 128

#define STACK\_SIZE 1500\*3

#define RANDOM\_PORT 1024 + rand() % 64511

## Data Structures

### TFTP Session

typedef struct

{

uint16 source\_port;

uint16 destination\_port;

uint32 source\_ip;

uint32 destination\_ip;

uint8 options;

uint8 timeout;

uint8 retry;

uint16 blksize;

uint8 windowsize;

} tftp\_session\_t;

Description: This structure is for storing data for each tftp session. timeout, retry, blksize, window size are initially set as default value. the are updated if options are received with a request. The options variable is used for indicating which options are requested by the client. Bit masking is used here.

### **Read write request**

typedef struct{

uint16 opcode;

char \*filename;

uint8 mode;

}rd\_wr\_request;

Description: The structure will store the value of the main part of the read write request. Based on the opcode value the further processing takes place.

### Data packet

typedef struct{

uint16 opcode;

uint16 block\_number;

char \*data;

} data\_packet;

Description: This structure is used for sending data packet to the client after each valid ACK reply and fisrt valid RRQ request.

### ACK packet

typedef struct{

uint16 opcode;

uint16 block\_number;

}ack;

Description: This packet will accept the ACK reply from the client when sending data and ACK response when sending data for each successful block transfer.

### OACK packet

typedef struct{

uint16 opcode;

uint8 options;

uint8 timeout;

uint8 retry;

uint16 blksize;

uint8 windowsize;

}oack;

Description: Based on the options variable the options are selected and there value are passed by the OACK. if options is zero then no OACK is sent.

### Error packet

typedef struct{

unit16 opcode;

unit16 error\_code;

char \* error\_msg;

};

## Main Global Variables

### TFTP Session Variable (example)

tftp\_session\_t tftp\_sessions[MAX\_TFTP\_SESSIONS];

Description: Store the data of running tftp sessions.

### Options for negotiation

enum options{

timeout = 1,

retry = 2,

blksize = 4,

windowsize = 8

};

Description: This options are supported by the tftp server. They have this types of value because they are used in bit-wise operation to set the value of “options” in tftp\_session\_t.

### Mode

enum mode{

OCTATE,

NETASCCI,

RAW

};

Description: This modes are numaric alternative of “octate”, “netascci”, “raw”. they are stored in the “mode” variable of structure rd\_wr\_request instate of string.

### Error

char errors[] ={

“Not defined, check error message”,

“File not found”,

“Access Violation”,

“Disk full or location”,

“Illegal TFTP operation”,

“Unknown transfer ID”,

“File already exist”,

“No such User”,

“Negotiation acknowledgement error”

};

Description: This global variable is created according to the error code. Here error code is taken as index and the error message as string. Only for index zero the error message will be anything based on the condition. But for the other error value the error message is fixed. And it can be set easily using the error code.

### semaphores

SEM\_ID global\_data\_locker;

Description: A binary semaphore used when changing the value of the global variable.

SEM\_ID reading\_file;

Description: A counting semaphore used when reading form a file. According to the requirement 3 concurrent read is possible. So the initial value of reading\_file semaphore is 3.

### Configurable variables

uint8 default\_timeout = 3;

uint8 default\_retry = 3;

uint16 default\_blksize = 512;

uint8 default\_windowsize = 1;

uint16 defaut\_port = 69;

Description: These value are used to store the default value of the tftp server variables. They can be configured by command and the configuration will overwrite the data with new data.

### Packet Type

enum opcode{

RRQ = 1,

WRQ,

DATA,

ACK,

ERROR,

OACK

};

Description: This variables are used to find the opcode received from the buffer and take decision.

# Procedures

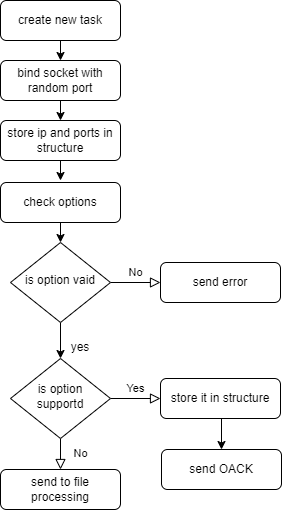
(Use flowchart or pseudocode to illustrate some important procedures in this module, but it is not necessary to provide all the detailed procedures.)

## Main Process of the Main Task

Pseudocode:

1. Initialize message queue and create main task..
2. Create a socket and bind it in default port of tftp.
3. register the socket with msg\_queue.
4. Wait forever for receiving message in the message queue.
5. Extract data from the buffer.
6. If vaild data received then forward it to the next component.
7. If not valid then send error and wait for next tftp request.

## Create a New Connection



# Interfaces

(List and describe the new interfaces between this module and other modules, as well as the main interfaces between the internal components of this module.)

## abc

Function: int abc(int a, int b, int c);

Description:

Arguments:

a:

b:

c:

Return value: