Design of tftpd

Revision History

Revision	Authors	Contact Info	Description	Date
1.0	Shanjid	BDCOM0061	Initial Design	12-02-2025

1. INTRODUCTION

1.1 Purpose

This document describes the design and implementation details of the TFTP (Trivial File Transfer Protocol) server for VxWorks. The server is designed to support file transfer operations within a **LAN environment**, focusing on **performance, reliability, and configurability**.

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.

1.2 References

1. RFC 1350 - The TFTP Protocol (Revision 2)
2. RFC 2347 – TFTP Option Extension
3. RFC 2348 - TFTP Blocksize Option
4. RFC 2349 – TFTP Timeout Interval & Transfer Size Options

1.3 Terminology

TFTP: Trivial File Transfer Protocol

RRQ: Read Request Packet

WRQ: Write Request Packet

ACK: Acknowledgment Packet

DATA: Data Packet

ERROR: Error Packet

OACK: Option Acknowledgement Packet

CLI: Command Line Interface

2. DESCRIPTION AND SPECIFICATIONS OF FUNCTIONALITIES

2.1 Background

TFTP is a simple file transfer protocol used in embedded systems and networking devices for bootstrapping, configuration loading, and remote file management. The server supports RRQ, WRQ, ACK, DATA, ERROR and OACK operations, ensuring adaptive timeouts, retransmission handling, and session management.

2.2 Configuration and Management Commands

2.2.1 Enable TFTP Server

Command: [no] tftp server enable
Configuration mode: Global configuration mode
Description: Enables or disables the TFTP server.
Parameters: None
Default: Enabled

2.2.2 Show the Configurations and Status

Command: show tftp server
Configuration mode: Privileged mode and global configuration mode
Description: Displays the current TFTP server settings and active sessions.
Parameters: None
Default: Not applicable

2.2.3 Show Running Configuration

Command: show running
Configuration mode: Privileged mode and global configuration mode
Description: Lists the configuration commands for the TFTP server (tftpd) that differ from the default values. This allows the operator to verify the current non-default configuration settings.
Parameters: None
Default: Not applicable

2.2.4 Show TFTP Server Version Information

Command: show version all or show version module tftpd
Configuration Mode: Privileged mode and global configuration mode
Description: Displays the version information for the TFTP server module (tftpd). This command helps in verifying the module version and any related version details during troubleshooting or maintenance.
Parameters: None
Default: Not applicable

2.2.5 Configurable UDP Port

Command: [no] tftp server port {port}
Configuration Mode: Global configuration mode
Description: Configures the UDP port on which the TFTP server listens. This command allows the operator to set a custom port if the default (typically port 69) is not desired.
Parameters:
{port}: A numeric value representing the desired UDP port.
Default: Port 69

2.2.6 Configurable Timeout and Retry Count for Retransmissions

Command: [no] tftp server retransmit {timeout} {retry}
Configuration Mode: Global configuration mode
Description: Sets the retransmission parameters for the TFTP server. The {timeout} parameter specifies the time (in seconds) the server waits before retransmitting a packet, while {retry} defines the number of transmission attempts (including the first normal transmission).
Parameters:
{timeout}: An integer value in the range 1–255 (inclusive) representing the timeout period in seconds.
{retry}: An integer value in the range 1–6 (inclusive) representing the total number of transmission attempts.
Constraint: The product of {timeout} and {retry} must not exceed 255.
Default:
Timeout: 3 seconds
Retry: 3 times

2.3 Functional Requirements

The TFTP server (tftpd) module is responsible for handling file transfers to and from the switch. It supports multiple client requests while ensuring controlled concurrency and configurable settings.

2.3.1 Core TFTP Server Functionality

- The TFTP server operates as an independent module within the switch software, enabling file uploads and downloads using the TFTP protocol.
- It supports **up to three concurrent read requests** from clients, even for the same file.
- **Simultaneous read and write sessions are not allowed**, nor are multiple concurrent write requests.

2.3.2 Configuration and Control

- The TFTP server can be enabled or disabled using the command:
  [no] tftp server enable
- The server's UDP port can be configured using:
  [no] tftp server port {*port*}
- Timeout and retry count for retransmissions are configurable with:

[no] tftp server retransmit {*timeout*} {*retry*}

- o Default timeout: **3 seconds** (Range: 1–255)
- o Default retry count: **3** (Range: 1–6)
- o The product of timeout and retry must not exceed **255**.

2.3.3 Feature Support and Monitoring

- The TFTP server supports RFC2348 blocksize negotiation.
- The system provides multiple CLI commands for monitoring and managing the TFTP server:
  - ◆ show tftp server – Displays configurations, active sessions, file names in operation, and ongoing operations (read/write).
  - ◆ show running – Lists configuration commands for non-default TFTP settings.
  - ◆ show version – Displays the version of the tftpd module using:
    - ▪ show version all
    - ▪ show version module tftpd
- Configuration settings can be saved into the startup configuration using the write command.

2.4 Performance Requirements

- Three concurrent read sessions should not exceed 125% of a single session's time.
- Efficient packet handling to minimize retransmissions.

3. ARCHITECTURE

The TFTP server module (tftpd) is a self-contained component integrated within the switch's software system. It provides file transfer services using the TFTP protocol and operates as a single task under VxWorks. The module interfaces directly with the operating system for network and file I/O operations and with the CLI subsystem for configuration and status monitoring.

3.1 TFTP Core and Session Management

The **TFTP Core and Session Management** component is responsible for the actual file transfer and protocol processing. It continuously monitors incoming UDP packets and processes them within the constraints of the session pool and concurrency limits.

3.2 CLI and Configuration Management

The **CLI and Configuration Management** component provides an operator interface for controlling the TFTP server. It communicates configuration changes and status queries to the TFTP Core, ensuring that the operational parameters (e.g., UDP port, timeout, and retry settings) are dynamically adjustable.

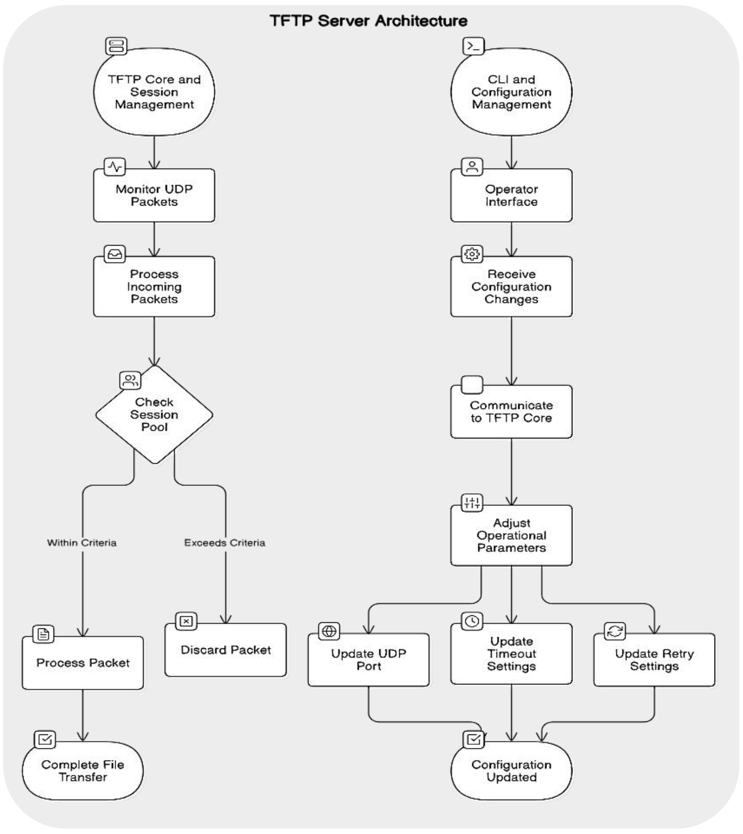


Figure 1: Architecture for TFTP Server module

4. DATA

4.1 Macro Definitions

4.1.1 Message Types

```
#define SOCKET_DATARCVD          0x9000100
#define SOCKET_CONNECTED         0x9000200
#define SOCKET_DISCONNECTING     0x9000300
#define SOCKET_DISCONNECTED      0x9000400
#define SOCKET_CANTSENDMORE      0x9000500
#define SOCKET_CANTRCVMORE       0x9000600
```

4.1.2 Packet Types

```
#define TFTP_RRQ 1
#define TFTP_WRQ 2
#define TFTP_DATA 3
#define TFTP_ACK 4
#define TFTP_ERROR 5
#define TFTP_OACK 6
```

4.1.3 Error Message Code

```
#define ERROR_UNKNOWN 0
#define ERROR_FILE_NOT_FOUND 1
#define ERROR_ACCESS_VIOLATION 2
#define ERROR_ALLOCATION_EXCEEDED 3
#define ERROR_ILLEGAL_OPERATION 4
#define ERROR_UNKNOWN_TID 5
#define ERROR_FILE_EXISTS 6
#define ERROR_NO_SUCH_USER 7
#define ERROR_OPTION_NEG_FAILED 8
#define ERROR_FILE_READ 9
#define ERROR_TIMEOUT 10
#define ERROR_SEND_FAILED 11
#define ERROR_RECV_FAILED 12
```

4.1.4 Default Values

```
#define OCTET_MODE 1
#define NETASCII_MODE 2
#define DEFAULT_TIMEOUT 3
#define MIN_BLKSIZE 8
#define DEFAULT_BLKSIZE 512
```

```
#define MAX_BLKSIZE 65464
#define DEFAULT_TSIZE 0
#define DEFAULT_RETRIES 3
#define MAX_TFTP_SESSIONS 3
#define MAX_FILENAME 256
#define DEFAULT_PORT 69
#define MIN_TIMEOUT 1
#define MAX_TIMEOUT 255
#define MIN_RETRIES 1
#define MAX_RETRIES 6
```

4.2 Data Structures

4.2.1 TFTP Session

```
typedef struct
{
    struct sockaddr_in client_addr;
    struct sockaddr_in server_addr;
    FCB_POINT *file_fd;
    char filename[MAX_FILENAME];
    char path[MAX_FILE_PATH + 1];
    uint32_t session_id;
    uint32_t block_counter;
    uint32_t file_offset;
    uint32_t bytes_transferred;
    uint16_t transfer_id;
    uint16_t opcode;
    tftp_options_t tftp_options;
    uint8_t mode;
    uint8_t options_enabled;
    uint8_t last_block;
    uint8_t active_status;
} tftp_session_t;
```

Description: This structure maintains the session state for a TFTP transfer. It holds information about the server and client addresses, file handling, transfer settings, and session statistics etc.

4.2.2 TFTP Server

```
typedef struct tftp_server_structure {
    int socket_fd;                      /* UDP socket for TFTP communications */
    uint16_t port;                      /* Configurable UDP port number */
    uint8_t retry_count;                /* Retry count (includes first transmission) */
    uint8_t session_count;              /* Count of active sessions */
    uint8_t active_write_session;       /* Flag indicating if any Active write session */
    uint8_t enabled;                    /* Flag indicating if TFTP server is enabled */
    tftp_options_t *p_tftp_options;     /* Points to Default or Current options */
    struct sockaddr_in server_addr      /* Info of Server IP, Port, Protocol etc. */
} tftp_server_t;
```

Description: The tftp_server_t structure represents the TFTP server instance, maintaining its configuration, state, and active session details. It includes the server's UDP socket, configurable port, retry count, and flags for tracking active sessions and write operations. Additionally, it holds a pointer to default or dynamically negotiated TFTP options and the server's network address information.

4.2.3 TFTP Options

```
typedef struct tftp_options_structure
{
    uint32_t timeout;
    uint32_t blocksize;
    uint32_t tsize;
} tftp_options_t;
```

Description: The tftp_options_t structure holds TFTP protocol options negotiated between the client and server. These options help configure transfer behavior, such as timeout duration, data block size, and file size.

4.2.4 ACK Packet

```
typedef struct tftp_ack_packet
{
    uint16_t opcode;
    uint16_t block_no;
} ack_packet_t;
```

Description: The Acknowledgment (ACK) packet is sent by both the client and the server to confirm the successful receipt of a DATA/OACK packet. It contains the opcode and the block number being acknowledged.

4.2.5 DATA Packet

```
typedef struct tftp_data_packet
{
    uint16_t opcode;
    uint16_t block_no;
    uint8_t data[0];
} data_packet_t;
```

Description: The DATA packet is used to transfer file data. It consists of an opcode, a block number, and a variable-length data field containing up to blocksize bytes of the file content.

4.2.6 OACK Packet

```
typedef struct tftp_oack_packet
{
    uint16_t opcode;
    uint8_t options[0];
} oack_packet_t;
```

Description: The Option Acknowledgment (OACK) packet is sent by the server in response to a client's read or write request when TFTP options (e.g., blocksize, timeout, tsize) are negotiated. It contains the opcode and a variable-length field listing the accepted options.

4.2.7 ERROR Packet

```
typedef struct tftp_err_packet
{
    uint16_t opcode;
    uint16_t error_code;
    char error_msg[0];
} error_packet_t;
```

Description: The ERROR packet is sent when an error occurs during transmission. It includes an opcode, an error code, and a variable-length error message describing the issue.

4.3 Main Global Variables

4.3.1 TFTP Session Variable

```
tftp_session_t tftp_sessions[MAX_TFTP_SESSIONS];
```

Description: An array that holds information for up to MAX_TFTP_SESSIONS concurrent TFTP sessions, including session state, client details, transfer progress, and file info.

4.3.2 File Transfer Semaphore

```
SEM_ID tftp_sem_sync, tftp_sem_lock;
```

Description: Semaphore used for synchronizing file transfer operations across multiple tasks as well as mutual exclusion ensuring proper access to shared resources like the buffer and file system.

4.3.3 Message Queue for Packet Receive

```
MSG_Q_ID tftp_msgQ_id;
```

Description: Message QueueID is to receive UDP socket packet which contains information like Socket Flag, Socket FD, Packet Length & User Argument passed from Socket Register.

5. PROCEDURES

The TFTP server module's implementation is based on a single-task, event-driven system running on a VxWorks-based switch. It leverages a UDP socket with message_queue (single block point) mechanisms to monitor incoming TFTP packets. Upon receiving a packet, the server determines whether to assign it to an existing session or to acquire a new session from a fixed-size pool.
Session management is central to the design. Up to three concurrent read (RRQ) sessions are allowed, while only one write (WRQ) session is permitted at any given time. For each session, the module maintains client information, file pointers, and transfer state, ensuring that sessions are properly initialized, processed, and released upon completion or error.
The server's operation is further enhanced by dynamic CLI-based configuration, enabling real-time adjustments to parameters such as the UDP port, retransmission timeout, and retry count. Status and version reporting commands provide visibility into active sessions and module details. This cohesive procedure guarantees reliable and efficient file transfers in a controlled, concurrent environment.

5.1 Main Process of the Main Task

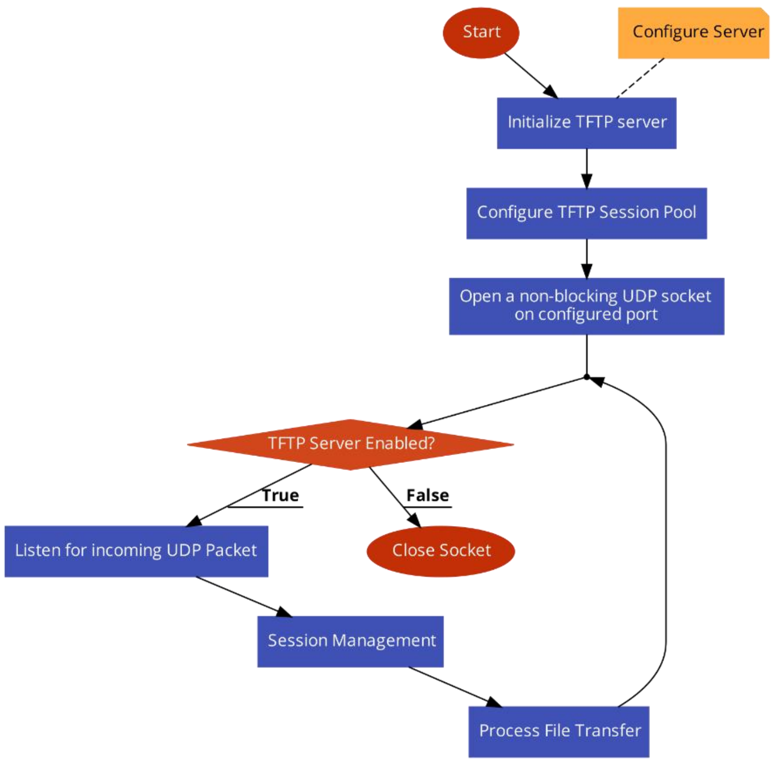


Figure 2: Flow chart of the main process of main task

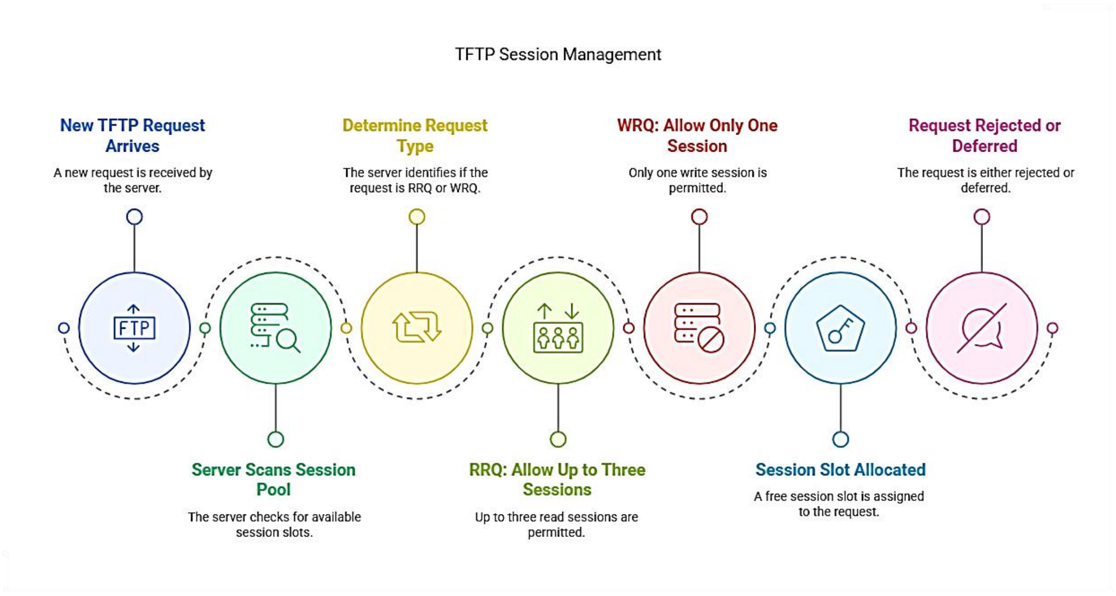
The TFTP server main task initiates by setting up the core server components and configuration. It begins by initializing the TFTP server and configuring a fixed session pool, ensuring resources are available for incoming transfers. A non-blocking UDP socket is then opened on the configured port, which facilitates asynchronous packet handling.

The task then enters a continuous loop that first checks if the TFTP server is enabled. If the server is disabled, it gracefully closes the socket and any other allocated resources. If enabled, the server listens for incoming UDP packets. Upon receiving a packet, the server engages its session management routines to either allocate a new session or associate the packet with an existing session.

Following session allocation, the file transfer process is executed according to the TFTP protocol—handling either read or write requests, managing retransmissions, and ensuring correct data block sequencing. Once the file transfer is processed, the loop reiterates, rechecking the server's enabled status and waiting for new packets, thus maintaining continuous operation and readiness to handle further client requests.

5.2 Session Management

5.2.1 Session Acquisition



TFTP Session Management
New TFTP Request Arrives
A new request is received by the server.
Server Scans Session Pool
The server checks for available session slots.
Determine Request Type
The server identifies if the request is RRQ or WRQ.
RRQ: Allow Up to Three Sessions
Up to three read sessions are permitted.
WRQ: Allow Only One Session
Only one write session is permitted.
Session Slot Allocated
A free session slot is assigned to the request.
Request Rejected or Deferred
The request is either rejected or deferred.
FTP

Figure 3: TFTP Session Acquisition Process

When a new TFTP request arrives, the server scans its fixed-size session pool to allocate a free session slot. For read requests (RRQ), it allows up to three concurrent sessions; for write requests (WRQ), it permits only one session, rejecting additional WRQ (and optionally any RRQ) while one is active. This mechanism ensures that each new client is only assigned a session if the concurrency constraints are met, and otherwise the request is rejected or deferred.

5.2.2 Session Processing

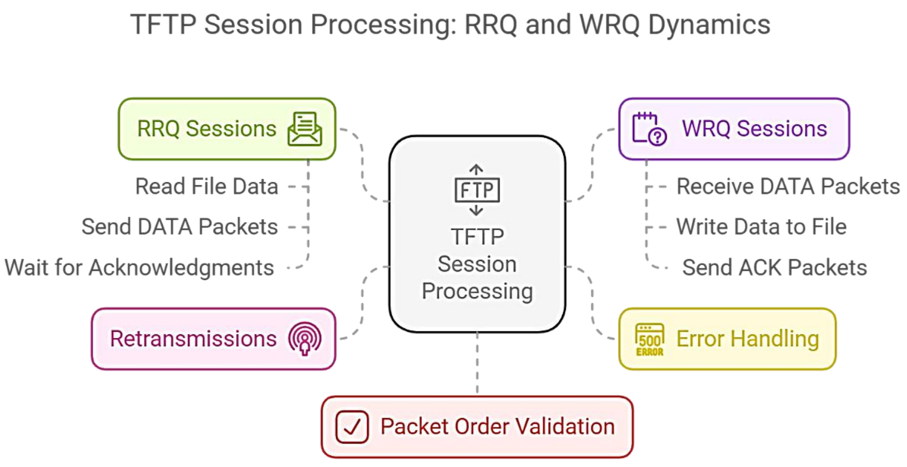


Figure 4: TFTP Session Processing with Data Transfer

Once a session is acquired, the server enters session processing, where it manages the data transfer according to the TFTP protocol. For RRQ sessions, the server reads file data and sends it as DATA packets, waiting for acknowledgments from the client. For WRQ sessions, it receives DATA packets from the client, writes the data to a file, and sends back ACK packets. This process includes handling retransmissions, validating packet order, and marking the session as complete when the transfer finishes or an error occurs.

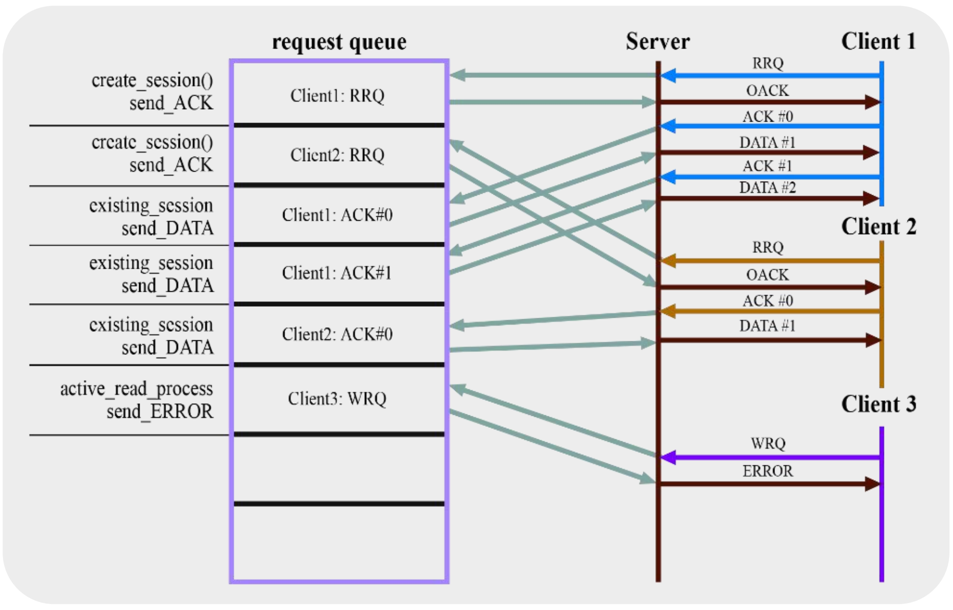
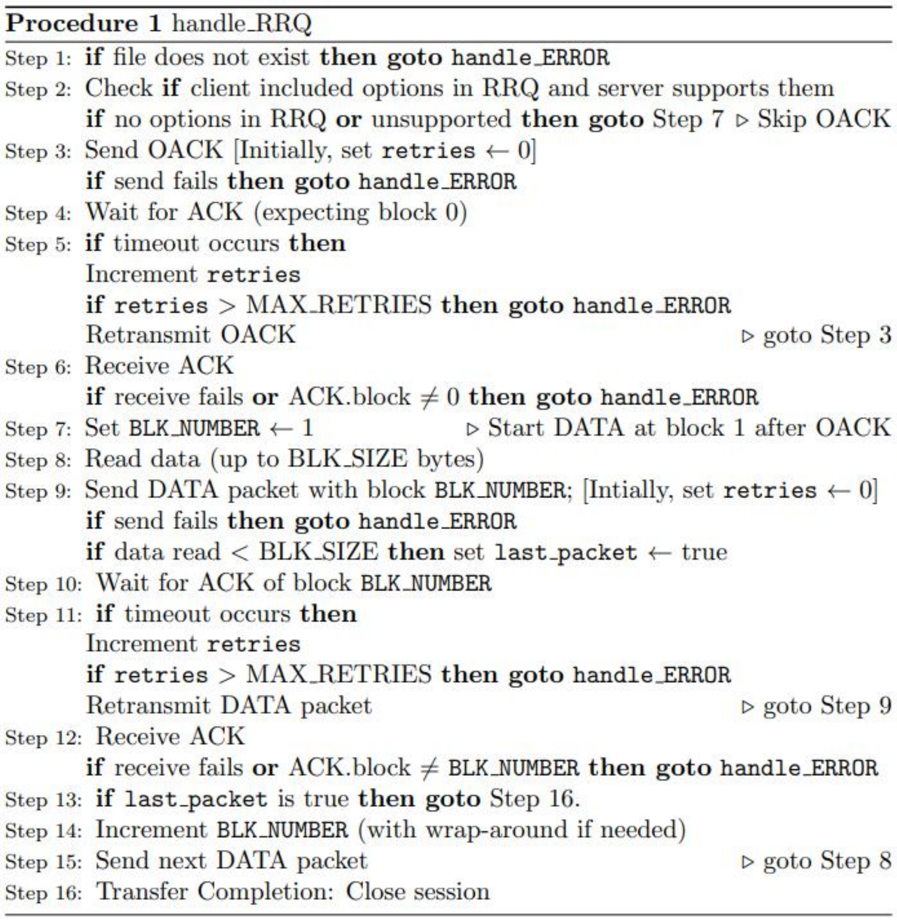


Figure 5: Demo workflow of the proposed TFTP module

5.3 Handle Read Request (RRQ)

This procedure manages a TFTP Read Request (RRQ) session. It begins by verifying the file's existence and checking for supported options. If options are present, an OACK (Option Acknowledgment) is sent, followed by waiting for an acknowledgment (ACK) from the client. After successful negotiation, data transfer starts with block numbering initialized at 1. Each data block is sent and awaits an ACK before proceeding. Retransmission mechanisms handle timeouts, ensuring reliability. The process continues until the last packet is sent and acknowledged, at which point the session is closed successfully.

Here is the procedural algorithm for handling RRQ:

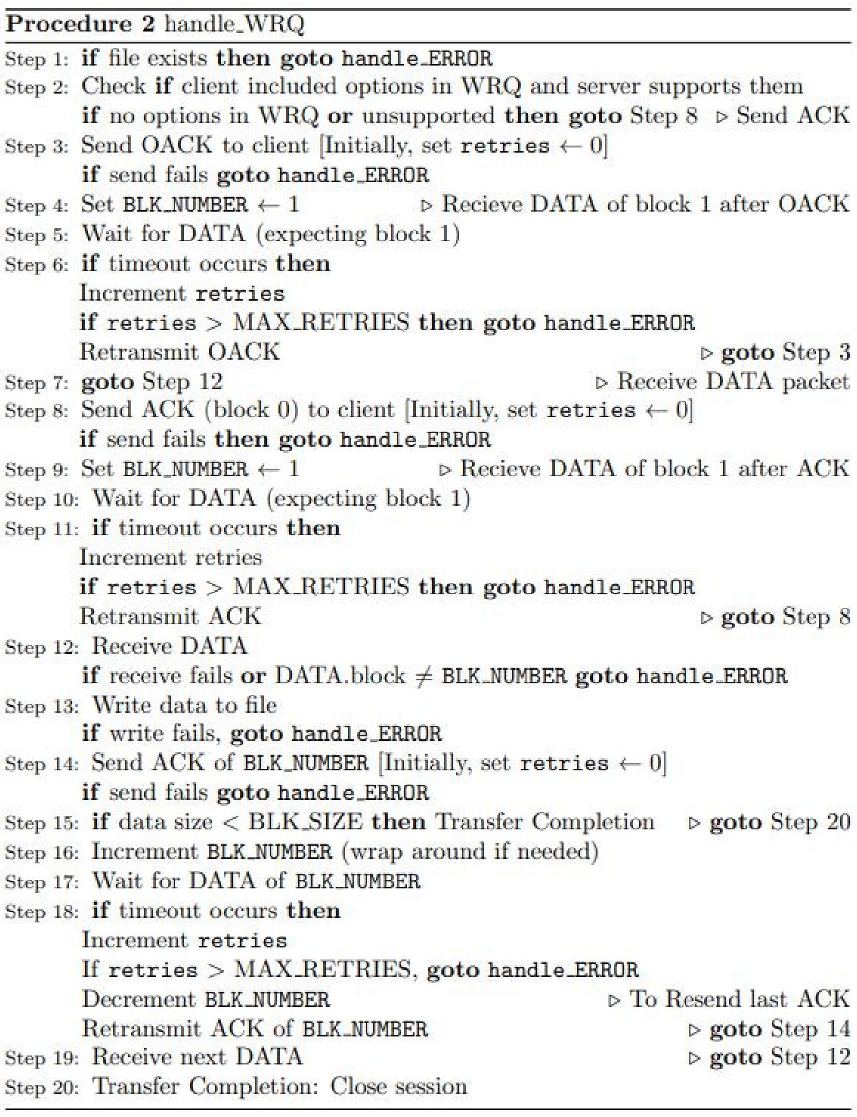


```
Procedure 1 handle_RRQ
Step 1:  if file does not exist then goto handle_ERROR
Step 2:  Check if client included options in RRQ and server supports them
         if no options in RRQ or unsupported then goto Step 7  ▷ Skip OACK
Step 3:  Send OACK [Initially, set retries ← 0]
         if send fails then goto handle_ERROR
Step 4:  Wait for ACK (expecting block 0)
Step 5:  if timeout occurs then
         Increment retries
         if retries > MAX_RETRIES then goto handle_ERROR
         Retransmit OACK                                  ▷ goto Step 3
Step 6:  Receive ACK
         if receive fails or ACK.block ≠ 0 then goto handle_ERROR
Step 7:  Set BLK_NUMBER ← 1              ▷ Start DATA at block 1 after OACK
Step 8:  Read data (up to BLK_SIZE bytes)
Step 9:  Send DATA packet with block BLK_NUMBER; [Intially, set retries ← 0]
         if send fails then goto handle_ERROR
         if data read < BLK_SIZE then set last_packet ← true
Step 10: Wait for ACK of block BLK_NUMBER
Step 11: if timeout occurs then
         Increment retries
         if retries > MAX_RETRIES then goto handle_ERROR
         Retransmit DATA packet                           ▷ goto Step 9
Step 12: Receive ACK
         if receive fails or ACK.block ≠ BLK_NUMBER then goto handle_ERROR
Step 13: if last_packet is true then goto Step 16.
Step 14: Increment BLK_NUMBER (with wrap-around if needed)
Step 15: Send next DATA packet                            ▷ goto Step 8
Step 16: Transfer Completion: Close session
```

5.4 Handle Write Request (WRQ)

This procedure manages a TFTP Write Request (WRQ) session. It begins by verifying that the target file does not already exist. If the client includes options, and they are supported, an OACK (Option Acknowledgment) is sent, followed by waiting for the first data block. If options are not used, an initial ACK is sent instead. The server then receives data blocks sequentially, writing them to the file and acknowledging each one. Retransmission mechanisms handle timeouts and ensure reliable data transfer. The process repeats until the final block is received and acknowledged, completing the transfer and closing the session.

Here is the detailed procedural algorithm for handling WRQ:



Procedure 2 handle_WRQ

```
Step 1:  if file exists then goto handle_ERROR
Step 2:  Check if client included options in WRQ and server supports them
         if no options in WRQ or unsupported then goto Step 8   ▷ Send ACK
Step 3:  Send OACK to client [Initially, set retries ← 0]
         if send fails goto handle_ERROR
Step 4:  Set BLK_NUMBER ← 1                  ▷ Recieve DATA of block 1 after OACK
Step 5:  Wait for DATA (expecting block 1)
Step 6:  if timeout occurs then
         Increment retries
         if retries > MAX_RETRIES then goto handle_ERROR
         Retransmit OACK                                      ▷ goto Step 3
Step 7:  goto Step 12                                 ▷ Receive DATA packet
Step 8:  Send ACK (block 0) to client [Initially, set retries ← 0]
         if send fails then goto handle_ERROR
Step 9:  Set BLK_NUMBER ← 1                   ▷ Recieve DATA of block 1 after ACK
Step 10: Wait for DATA (expecting block 1)
Step 11: if timeout occurs then
         Increment retries
         if retries > MAX_RETRIES then goto handle_ERROR
         Retransmit ACK                                       ▷ goto Step 8
Step 12: Receive DATA
         if receive fails or DATA.block ≠ BLK_NUMBER goto handle_ERROR
Step 13: Write data to file
         if write fails, goto handle_ERROR
Step 14: Send ACK of BLK_NUMBER [Initially, set retries ← 0]
         if send fails goto handle_ERROR
Step 15: if data size < BLK_SIZE then Transfer Completion     ▷ goto Step 20
Step 16: Increment BLK_NUMBER (wrap around if needed)
Step 17: Wait for DATA of BLK_NUMBER
Step 18: if timeout occurs then
         Increment retries
         If retries > MAX_RETRIES, goto handle_ERROR
         Decrement BLK_NUMBER                            ▷ To Resend last ACK
         Retransmit ACK of BLK_NUMBER                         ▷ goto Step 14
Step 19: Receive next DATA                                    ▷ goto Step 12
Step 20: Transfer Completion: Close session
```

6. INTERFACES

This section lists and describes the interfaces between the TFTP server module and other modules, as well as the main internal interfaces between components within the TFTP server module.

6.1 tftpd_session_acquisition

Function: p_tftp_session_t tftpd_session_acquisition(char *buff_ptr, int length);
Description: Parses incoming TFTP packets to extract the relevant information and create a session if needed. This function identifies the type of TFTP request and returns a session object that holds session details like file names, client information, and packet status.
Arguments:
  buff_ptr: A pointer to the received data buffer that contains the TFTP packet.
  length: The length of the data in the buffer.
Return value:
  A pointer to the tftp_session_t structure: session created or found
  Returns NULL: if the packet is invalid or cannot be parsed.

6.2 tftpd_handle_RRQ

Function: int tftpd_handle_RRQ(p_tftp_session_t session);
Description: Handles a TFTP read request from a client. This function processes the read request, determines the file to be sent, and prepares for sending the data to the client. It ensures that the requested file is available and handles reading the data correctly.
Arguments:
  session: A pointer to the current TFTP session object that contains session-specific details like client IP, port, and file name to be read.
Return value:
  SESSION_SUCCESS on success or SESSION_FAILURE indicate an error.

6.3 tftpd_handle_WRQ

Function: int tftpd_handle_WRQ(p_tftp_session_t session);
Description: Handles a TFTP write request from a client. This function processes a write request, manages the file transfer process, and prepares for subsequent data reception from the client. It ensures that the server writes data to the correct file and handles acknowledgment appropriately.
Arguments:
  session: A pointer to the current TFTP session object that contains session-specific details like client IP, port, and file information.
Return value:
  SESSION_SUCCESS on success or SESSION_FAILURE indicate an error.