Part 1: A Simplified Implementation of the Trivial File Transfer Protocol in Java

INTRODUCTION

Trivial File Transfer Protocol (TFTP) is a simple file transfer protocol that runs on top of User Datagram Protocol (UDP). TFTP is often used for transferring small files between devices, such as configuration files or firmware updates. This report discusses an implementation of a simplified TFTP over UDP in Java.

TFTP PROTOCOL OVERVIEW

TFTP uses UDP as the underlying transport protocol. The TFTP protocol includes two types of messages: Request (RRQ/WRQ) messages and Data messages. The request messages are used to initiate file transfers and the data messages are used to transfer the file data.

RTF's specification of TFTP has two modes of operation: netascii mode and octet mode. In netascii mode, files are transferred as ASCII text with line terminators converted to the appropriate format for the destination system. In octet mode, files are transferred as binary data. The Request datagram packets used contain an opcode (2 bytes), a filename (string), a zero (1 byte) to indicate the end of filename, a mode (string), and a final zero (1 byte) to signal the end of the string.

This simplified TFTP has only one mode of operation which is octet mode. This results in a reduced Request packets because there is no need to send the mode. The resulting Request packet will only contain the opcode (2 bytes) followed by the filename (string). The data is sent in an array of bytes (8-bits). The string will be converted to byte form and sent together with the opcode. Thus, the filename can be determined by the data following an RRQ/WRQ opcode running until the end of the packet.

IMPLEMENTATION OF TFTP IN JAVA

To implement TFTP over UDP in Java with a server that support simultaneous file transfers to and from multiple clients, three classes will be created: TFTPClient, TFTPServer, and TFTPServerThread. The TFTPClient class will be responsible for sending requests and receiving data, while the TFTPServer class will receive request packets from clients and pass them to TFTPServerThread which responds to the requests and sends data.

This was implemented as follows:

TFTPServer

The server is multi-threaded, meaning it can handle multiple file transfers simultaneously. Each file transfer is handled by a separate thread, which extends the Thread class. The main function of this program is to receive request packets from clients and pass them to a TFTPServerThread which responds to read and write requests (RRQ/WRQ). The program uses port number 1234 to avoid issues of administrator's rights, and it only supports octet

mode. The program runs in an infinite loop, waiting for incoming packets from clients. If an invalid opcode is received, it outputs an error message to the console.

The program creates a DatagramSocket on the TFTP_PORT and listens for incoming packets from clients. When a packet is received, it extracts the opcode from the packet and checks if it is a valid request opcode. If the opcode is valid, it creates a new thread to process the received packet using the TFTPServerThread class.

TFTPServerThread

The TFTPServerThread class handles the details of sending and receiving data according to the TFTP protocol and communicates with the client using DatagramSocket to send and receive DatagramPackets.

A Server Thread creates a DatagramSocket with any available port and waits for incoming DatagramPacket requests from clients. The incoming packet is copied to a local buffer, and the filename requested is extracted. If the opcode in the buffer indicates a Read Request (OP_RRQ) or a Write Request (OP_WRQ), the corresponding method is called to process the request.

If a Read Request is received, the server checks if the requested file exists, and if it does, it opens the file for reading and starts sending data to the client in packets. Each packet contains a 2-byte opcode, a 2-byte block number, and up to 512 bytes of data. After sending each packet, the server waits for an Acknowledgment (ACK) packet from the client. If the ACK is not received within the specified timeout, the server resends the packet. If the server receives an error packet instead of an ACK, it prints an error message and stops the transfer. When the end of the file is reached, the server sends a short packet to indicate the end of the transfer.

If a Write Request is received, the server opens the requested file for writing and waits for incoming data packets from the client. Each data packet contains a 2-byte opcode, a 2-byte block number, and up to 512 bytes of data. After receiving each packet, the server sends an ACK packet to the client to confirm that the packet was received. If the server receives an error packet instead of a data packet, it prints an error message and stops the transfer.

TFTPClient

The TFTP client is used to send read and write requests to a server. It uses the DatagramSocket class to send and receive datagrams and supports only octet mode.

The code has several methods to support its functionality, including a readRequest method that sends a read request to the server, reads the packets sent in response, and writes them to a file. The writeRequest method sends a write request to the server and sends packets containing the file data to the server.

The main method checks that the correct number of input arguments is passed and creates a new TFTP client. The run method prompts the user for an instruction and a filename and calls the readRequest or writeRequest method based on the instruction entered.

The TFTPClient class also has several fields, including an InetAddress to store the address of the server, a DatagramSocket to create a socket for sending and receiving datagrams, and DatagramPackets to store the datagrams sent and received. It also has constants for the opcodes used in TFTP (OP_RRQ, OP_WRQ, OP_DATA, OP_ACK, and OP_ERROR) and for the packet size and timeout.

Part 2: An Implementation of TFTP Over TFP in Java

IMPLEMENTATION

The implementation of TFTP over TCP in Java involves creating two programs: a server and a client. The server listens on a specific port and waits for client connections. When a client connects, the server creates a new thread to handle the connection. The client, on the other hand, initiates the connection and sends requests to the server.

TftpTCPClient

The TftpTCPClient class provides a main method that creates a socket connection to the server and uses DataOutputStream and BufferedReader objects to send and receive data over the socket. It also uses a BufferedReader to read user input from the command line and FileReader and BufferedWriter objects to read from and write to files.

The client first takes two command line arguments, the server address and the port number, and then prompts the user to enter a request and a filename. The request is a two-character code that specifies whether the client wants to read from or write to the server, and the filename is the name of the file to be transferred.

If the request is to read from the server, the client sends a request packet to the server with the opcode and filename. The server responds with a data packet that contains the contents of the file, which the client reads and writes to a local file. If no data is received from the server, the client deletes the local file.

If the request is to write to the server, the client reads the contents of the local file and sends them to the server in data packets. Once all the data has been sent, the client closes the socket connection. The implementation includes error handling for cases such as invalid command line arguments, invalid opcodes, and file not found errors.

TftpTcpServer

The `main()` method creates a `ServerSocket` object on port number 10000 and listens for incoming client connections. When a new client connects, it accepts the connection and creates a new `ClientHandler` thread to handle the client's requests. The `ClientHandler` class is responsible for implementing the TFTP protocol on top of TCP for the client.

The `while` loop in the `main()` method runs indefinitely, accepting incoming client connections one after the other. It waits until a new client connects and then creates a new thread to handle the client's requests. This allows the server to handle multiple clients simultaneously.

ClientHandler

The `ClientHandler` constructor takes a `Socket` object as a parameter, which represents the client's socket connection to the server.

The `run()` method overrides the `run()` method of the `Thread` class. It first sets a timeout of 5 seconds on the client socket. It then creates `DataOutputStream` and `BufferedReader` objects for writing to and reading from the socket, respectively.

The method then reads the client's request from the socket using the `BufferedReader` object. The request is parsed to determine the TFTP operation code (RRQ or WRQ) and the filename. The method then checks whether the operation code is valid and proceeds to handle the request accordingly.

If the operation code is RRQ (read request) and a filename has been provided, the method opens the file with the given filename and reads its contents. The contents are then sent back to the client using the `DataOutputStream` object.

If the operation code is WRQ (write request) and a filename has been provided, the method creates a new file with the given filename and writes the data received from the client to the file. If no data is received, the file is deleted.

CONCLUSION

This report has discussed the implementation of TFTP over TCP in Java. TFTP is a simple file transfer protocol that is widely used in local networks. Implementing TFTP over TCP in Java involves defining the format of the TFTP messages and the protocol for their exchange. The implementation can be used to transfer files between two networked devices in a reliable and efficient manner.