# 1DV701 Assignment 3 - TFTP Server Report

**Contribution:** [Hirendhar Ramkumar (hr222rs)]: 100% (Implemented all server logic, error handling, testing, and report writing)

I developed a TFTP server in Java per RFC1350, supporting "octet" mode for read (RRQ) and write (WRQ) requests over UDP on port 1234. For Problem 1, I extended the provided starter code to handle single read and write requests for files under 512 bytes. I implemented receiveFrom() to listen for client packets, parseRequest() to extract opcode, filename, and mode, and logic to send DATA packets for RRQ or receive DATA for WRQ. Testing with a TFTP client on localhost:1234 confirmed successful transfers . The server uses two sockets: a main socket on port 1234 to listen for requests and a per-client socket for transfers, ensuring concurrency without blocking new connections. My approach involved iterative development, debugging with console logs, and referencing RFC1350 for packet structures.

For VG-Problem 2, I enabled large file transfers and timeouts. I modified sendDataReceiveAck() for RRQ to send 512-byte DATA packets with incrementing block numbers, stopping at a < 512-byte packet, and receiveDataSendAck() for WRQ to write DATA packets, sending ACKs per block. I added a 2-second timeout and 5 retries for RRQ ACKs and WRQ DATA packets, halting retransmissions to avoid infinite loops. Testing involved transferring large files (e.g., 1MB PDF) and simulating packet loss to verify retransmissions

For VG-Problem 3, I analyzed a read request using Wireshark, capturing RRQ (opcode 1, filename, "octet"), DATA (opcode 3, block number, 512 bytes), ACK (opcode 4, block number), and a final < 512-byte DATA packet . Read requests involve server-to-client data flow, while write requests reverse this, starting with a server ACK (block 0) followed by client DATA packets.

For VG-Problem 4, I implemented error handling for codes 0 (invalid extension: "Invalid file type"; size exceeded: "File exceeds size limit"), 1 ("File not found"), 2 ("Access violation"), and 6 ("File already exists"). I added file validation for WRQ, restricting extensions to .txt, .pdf, .doc, .docx, .jpg, .png, .ul and enforcing a 10MB limit. Error packets (opcode 5) were sent via sendError(), and incoming error packets terminated connections. Testing covered error scenarios. My solution meets RFC1350, with robust testing ensuring reliability.