Computer Networks Report

**How the code runs**

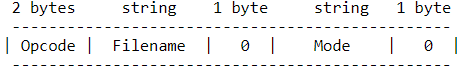
Initial stage of the project involved understanding how a UDP protocol worked using a client server relationship. Lab work on UDP socket programming provided the foundations for my TFTP to run over UDP. The protocol works as follows; a server is instantiated by calling the main function in class TFTPUDPServer. The servers port is set to 2044 due to validation issues with port 69. Servers while function causes it to run indefinitely until closed by the user.

When the class TFTPUDPClient’s main function is called a new client is instantiated which calls onto my commandArgs method. This method implements a scanner to read input from the console line. String messages are passed to the user asking them to enter an IP address, file name and read or write request. Client fields are updated by the console input to set IP address and file we want to access. A switch statement differentiates between a read and write request. Case write request writes the file in client to an empty byte array called insert to be sent to the server to be written. A ByteArrayOutputStream has a request block placed into it by calling function generateRequestBlock which takes an opcode either 1 for read request or 2 for write request and outputs the necessary format for a request packet as specified by the RFC1350 . This datagrampacket is then sent to the server on port 2044 using the IP address inputted in the console line and the request packet generated by inputting 1 or 2 for rrq or wrq. The socket is given a 4 second timer after the initial packet is sent to receive something back or else it times out and exits.

Once a packet is sent to the server a server thread is instantiated and the packet is sent to the thread. When the server thread is instantiated a random port number is passed to it above 1024. The run method in serverThread takes a packet and extracts its op code. I implemented a switch statement for different cases dependent on the opcode received from the packet. If a read request is received then a data packet is sent back to the client. If a write request is received an ack is sent back to client. If an ack is received then data is sent back. If data is received an ack is sent back. Receiving last packet follows with a reply to client sending an empty ack. Upon exiting the switch cases write request is evaluated. If the packet received was a write request then the file is written to the server directory. Data is passed back an forth between the serverThread and Client using ‘octet’ which refers to raw sequences of bytes. If read request is received data is sent to the client and closes once all of the data has been sent. If write request is received acks are sent and data is passed back in. The client exits once all data has been sent but server stays running until stopped manually.

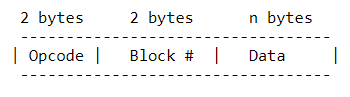
RFC1350

Implementing a TFTPUDP protocol required the help of the RFC1350 – the classification document explaining how the protocol works. From this document I ascertained that the main functionality of my program was to send text files as raw sequences of bytes using ‘octet’ mode. A working read and write request were essential to the protocol’s success. According to the specification request packets implement the following structure.



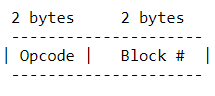
My method in client generateRequestBlock takes as an input, the opcode of either a read or write request dependent on what the user specifies in the console line args and returns a byte array with the entire request inside. The byte array in this method containsRqst is given size corresponding to the filenames length and the modes (in this case octet) length with the first two bytes assigned to opcode and 1 byte separating file and mode.

According to the RFC data packets require the following structure in order to be implemented correctly.



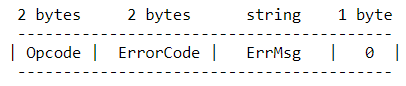
Data packets are written in both client on write request and written in server thread when a read request has been initialised. The length of each data packet is declared as a field with size 516 corresponding to the 4 bytes required for opcode and block and then the maximum amount of bytes data can take being 512 before a new packet must be made. Both serverThread and Client call on method sendCreatedData which wraps data information and a block number into one packet and sends it over the socket.

RFC also specifies that ACK packets take the following structure.



Ack packets are written in both client on read request (replying to data being received) or written in serverThread on write request. Both classes have a method sendCreatedAck which takes a blocknumber and places it into a byte array using opcode and block number structure seen above. The packet is then sent over the socket.

RFC finally specifies that error packets need the following structure



An error is sent to the client only when no such file can be found upon receiving a read request. A method sendCreatedError takes input of a message you want to send in the error. The opcode and error code are written into a temp byte array which is wrapped into the final error with the string message and is finally sent over the socket. Our assignment has specified that our program should only support error handling when a file cannot be found. This is the reason why error code is set to 1 as this is the only error we are interested in.

The code written is successful writing and reading files to and from the server on the loopback address 127.0.0.1. When trying to implement interoperability errors were throw not allowing packets to be sent over the internet. At this point I am not sure if it is a validation issues from public wifi or an error in the codes logic. Due to other external time constraints I did not have the means to fix this but would like to in the future. Text file file.txt is held in server package and can written to client package by inputting 127.0.0.1 followed by file.txt and finally 1 for rrq. The file is empty but has been tested with data inside to verify the programs functionality.