Balcueva, J. Escalona, J.M.

Fadrigo, J.A.M. Fortiz, P.R.

NSCOM01

TFTP Client - Program Design

Program Specifications

1. Program Language: Java
2. Interface: GUI
3. Target Features:
    a. Key Features
        i. GUI or a command line-based user interface are acceptable.
        ii. The user is allowed to specify the server IP address.
        iii. Support for both upload and download of binary files
        iv. When uploading, the program can send any file on the computer to the TFTP server as long as the file is accessible to the user using his / her OS privileges
        v. When downloading, the program must allow the user to provide the filename to use when saving the downloaded file to the client's computer
        vi. Proper error handling at the minimum should include the following:
            1. Timeout for unresponsive server
            2. Handling of duplicate ACK
            3. User prompt for file not found, access violation, and disk full errors
    b. Optional Features
        i. Support for option negotiation will merit additional points if correctly implemented
            1. Option to specify the transfer block size
            2. Communicate transfer size to a server when uploading
        ii. To allow the user to manually ping the target host prior to transmission.

Key Classes List

1. Client - Conducts the file reading/writing and transmission of TFTP packets.
2. TFTP - Builds the TFTP packets that will be used in TFTP transmission. The class also includes methods for checking and validating packet parameters that can affect the packet’s overall validity and usability in the TFTP protocol.

TFTP Packet Diagram

The design of the packets was heavily referenced from the samples from the RFC documents and actual TFTP packets using TFTPd64 and Wireshark. The resulting packet tables are as follows. The table shows both ways of appending a padding byte for strings where if the string (as per ASCII specification) does have a 0\, then it does not need a padding byte. The opposite goes if it does have \0.

1. REQUEST (READ/WRITE, W/O OPTVALS)

REQUEST (READ/WRITE, W/O OPTVALS)						
Length	2 Bytes		Length of String	1 Byte	Length of String	1 Byte
Segment	PADDING 0	Type 01/02	Filename (assuming without \0)	PADDING 0	Mode	PADDING 0

2. REQUEST (READ/WRITE, W OPTVALS)

REQUEST (READ/WRITE, W OPTVALS)											
Length	2 Bytes		Length of String	1 Byte	Length of String	1 Byte	Length of String	Length of String	...	Length of String	Length of String
Segment	PADDING 0	Type 01/02	Filename (assuming without \0)	PADDING 0	Mode	PADDING 0	Opt1 (assuming w/ \0)	Val1 (assuming w/ \0)	...	OptN (assuming w/ \0)	ValN (assuming w/ \0)

3. DATA

DATA			
Length	2 Bytes		Length of Byte[]
Segment	PADDING 0	Type 01/02	Byte[] Data

4. ACK

ACK				
Length	2 Bytes		2 Bytes	
Segment	PADDING 0	Type 01/02	PADDING 0	Block #

5. OACK

OACK						
Length	2 Bytes		Length of String	Length of String	...	Length of String
Segment	PADDING 0	Type 01/02	Opt1 (assuming w/ \0)	Val1 (assuming w/ \0)	...	OptN (assuming w/ \0)

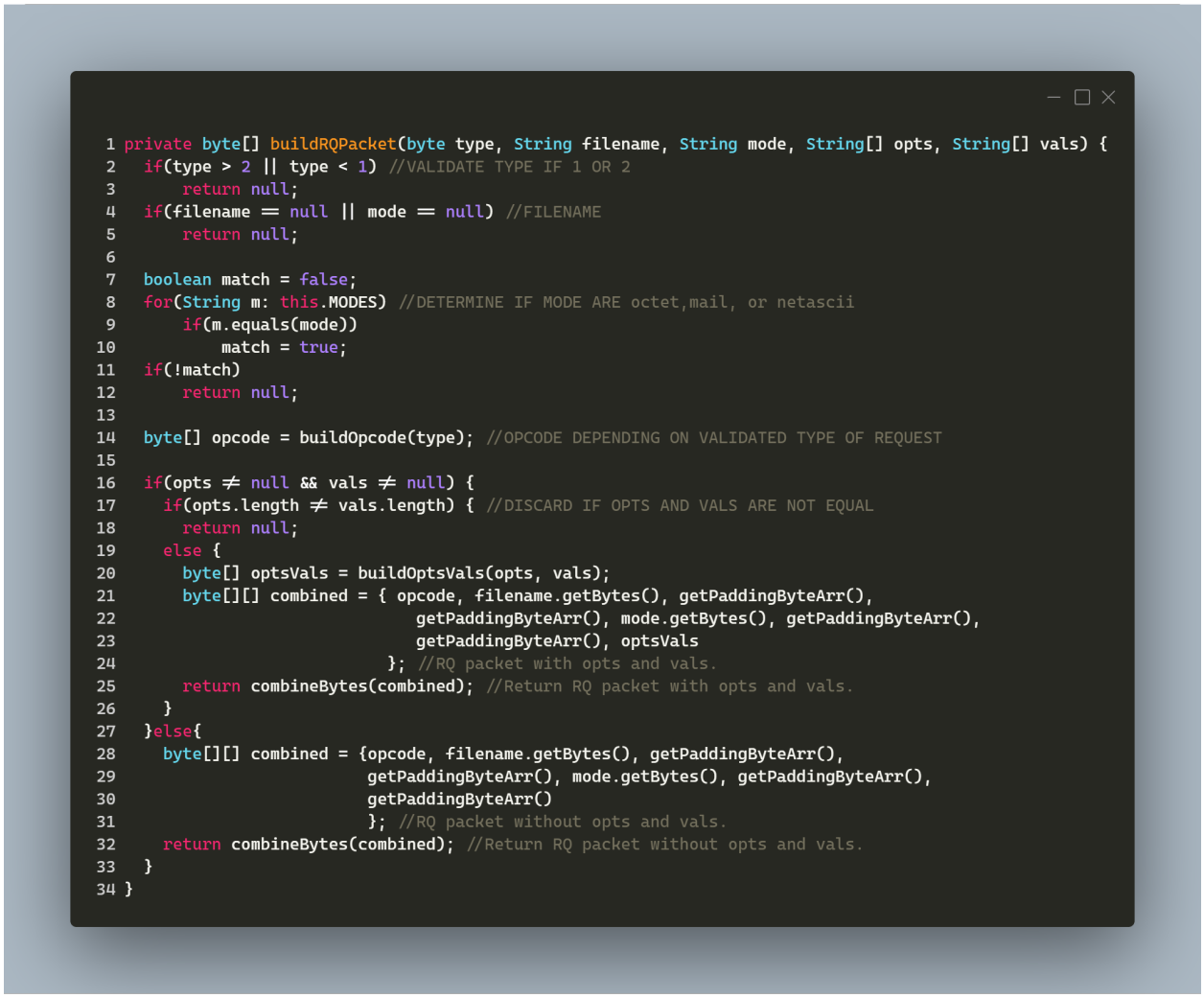
6. ERROR PACKET

ERROR PACKET					
Length	2 Bytes		2 Bytes		1 Byte
Segment	PADDING 0	Type 01/02	PADDING 0	Error Message (assuming w/ \0	PADDING 0

TFTP Packet Assembly Code

This segment contains the functions used in assembling TFTP packets that will be used in the TFTP transmission. Some of the functions are abstracted from the code shown. The contents of the abstracted methods/functions are found in the appendix below. The method parameters are also pre-validated before methods are called.

1. Request Packet (w/ or w/o OptVals, and whether Read=1 or Write=2)



```
private byte[] buildRQPacket(byte type, String filename, String mode, String[] opts, String[] vals) {
  if(type > 2 || type < 1) //VALIDATE TYPE IF 1 OR 2
      return null;
  if(filename == null || mode == null) //FILENAME
      return null;

  boolean match = false;
  for(String m: this.MODES) //DETERMINE IF MODE ARE octet,mail, or netascii
      if(m.equals(mode))
          match = true;
  if(!match)
      return null;

  byte[] opcode = buildOpcode(type); //OPCODE DEPENDING ON VALIDATED TYPE OF REQUEST

  if(opts ≠ null && vals ≠ null) {
    if(opts.length ≠ vals.length) { //DISCARD IF OPTS AND VALS ARE NOT EQUAL
      return null;
    else {
      byte[] optsVals = buildOptsVals(opts, vals);
      byte[][] combined = { opcode, filename.getBytes(), getPaddingByteArr(),
                            getPaddingByteArr(), mode.getBytes(), getPaddingByteArr(),
                            getPaddingByteArr(), optsVals
                          }; //RQ packet with opts and vals.
      return combineBytes(combined); //Return RQ packet with opts and vals.
    }
  }else{
    byte[][] combined = {opcode, filename.getBytes(), getPaddingByteArr(),
                         getPaddingByteArr(), mode.getBytes(), getPaddingByteArr(),
                         getPaddingByteArr()
                         }; //RQ packet without opts and vals.
    return combineBytes(combined); //Return RQ packet without opts and vals.
  }
}
```

2. OACK Packet

```java
private byte[] buildOACKPacket(String[] opts, String[] vals) {
  if(opts == null || vals == null) //CHECK IF OPTS AND VALS ARE NULL
    return null;
  if(opts.length != vals.length) //CHECK IF OPTS AND VALS ARE NOT EQUAL
    return null;
  byte opcodeVal = 6; //OPCODE AS 6
  byte[] optCode = buildOpcode(opcodeVal); //OPCODE AS BYTE
  byte[] combinedOptsVals = buildOptsVals(opts, vals); //COMBINE OPTS & VALS AS ONE BYTE[]
  byte[][] combined = {optCode, combinedOptsVals}; //COMBINE AS COMPLETE OACK PACKET
  return combineBytes(combined);
}
```

3. ACK Packet

```java
private byte[] buildACKPacket(Short block) {
  if(block < 0) //DISCARD IF BLOCK IS NEGATIVE
    return null;
  byte opcode = 4; //OPCODE AS 4
  byte[][] combined = { buildOpcode(opcode),
                        getPaddingByteArr(),
                        u.shortToByteArr(block)
                      }; //COMBINE AS ACK PACKET
  byte[] ack = combineBytes(combined);
  return ack;
}
```

4. Data Packet

```java
private byte[] buildDataPacket(Integer block, byte[] data) {
  if(block < 0) //DISCARD IF BLOCK IS NEGATIVE
    return null;
  if(data == null) //DISCARD IF THERE IS NO DATA
    return null;

  byte opcodeVal = 3; //OPCODE AS 3
  Short blockShort = block.shortValue(); //BLOCK#

  byte[] opcode = buildOpcode(opcodeVal) //OPCODE AS BYTE[]
  byte[] blockNum = {getPaddingByte(),blockShort.byteValue()}; //BLOCK# AS BYTE[]
  byte[][] preDataPacket = {    opcode, getPaddingByteArr(),
                                getPaddingByteArr(), blockNum,
                                data
                           }; //COMBINE AS DATA PACKET
  return combineBytes(preDataPacket);
}
```

5. ERROR Packet

```java
private byte[] buildErrPacket(Integer err, String emsg) {
  if(emsg.charAt(emsg.length()-1) != '\0') //CHECK IF EMSG DOES NOT HAVE \0
    emsg += '\0';

  if(!validErrCode(err)) //CHECK IF err IS A VALID ERROR CODE
    return null;

  byte opcodeVal = 5; //OPCODE AS 5

  byte[] opcode = buildOpcode(opcodeVal)
  byte[] errcode = {    getPaddingByte(), getPaddingByte(), getPaddingByte(),
                        err.byteValue()
                   }; //ERROR CODE ASSEMBLY
  byte[] errMsg = emsg.getBytes(); //ERROR MSG AS BYTES
  byte[][] combined = { opcode, errcode, errMsg,
                        getPaddingByteArr(), getPaddingByteArr(),
                        getPaddingByteArr()
                      }; //COMBINE AS ERROR PACKET
  return combineBytes(combined);
}
```

TFTP Packet Assembly Results and Reference

From the packet assembly code shown above, the resulting outputs (parsed to hex and bits) alongside the Wireshark packet reference are as follows.

1. Error Packet

```
Error Packet
System:
System Hex from Processed Byte: 0005000146696c65206e6f7420666f756e640000
System Bits: 00000000 00000101 00000000 00000001 01000110 01101001 01101100 01100
101 00100000 01101110 01101111 01110100 00100000 01100110 01101111 01110101 01101
110 01100100 00000000 00000000
Wireshark:
Wireshark Hex Raw: 0005000146696c65206e6f7420666f756e640000
Wireshark Bits: 00000000 00000101 00000000 00000001 01000110 01101001 01101100 01
100101 00100000 01101110 01101111 01110100 00100000 01100110 01101111 01110101 01
101110 01100100 00000000 00000000
isError: true
Extract Error: 1 = File not found
```

```
Trivial File Transfer Protocol
    Opcode: Error Code (5)
    [Destination File: nenechi.png]
    [Read Request in frame 425]
    Error code: File not found (1)
    Error message: File not found
  > [Expert Info (Warning/Response): TFTP ERROR packet]
```

```
0000  10 63 c8 5f 57 11 30 9c  23 63 6f c3 08 00 45 00   ·c·_W·0· #co···E·
0010  00 30 62 93 00 00 80 11  24 e0 c0 a8 18 fd c0 a8   ·0b····· $·······
0020  18 fc f1 3a c4 26 00 1c  56 13 00 05 00 01 46 69   ···:·&·· V·····Fi
0030  6c 65 20 6e 6f 74 20 66  6f 75 6e 64 00 00         le not f ound··
```

2. Data Packet

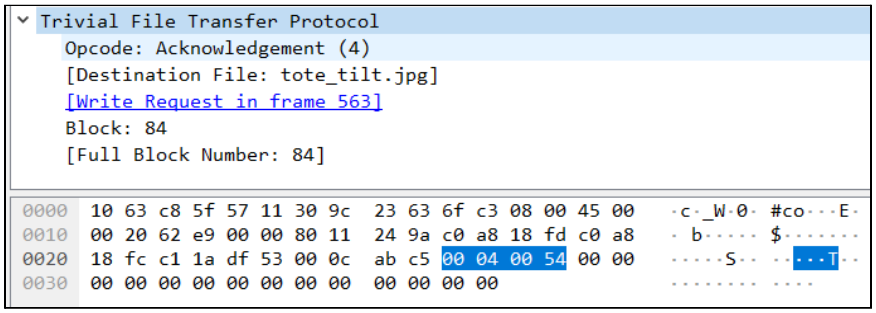
```
Data Packet
System:
System Hex from Processed Byte: 0003000168656c6c6f20776f726c64
System Bits: 00000000 00000011 00000000 00000001 01101000 01100101 01101100 01101
100 01101111 00100000 01110111 01101111 01110010 01101100 01100100
Wireshark:
Wireshark Hex Raw: 0003000168656c6c6f20776f726c64
Wireshark Bits: 00000000 00000011 00000000 00000001 01101000 01100101 01101100 01
101100 01101111 00100000 01110111 01101111 01110010 01101100 01100100
getOpCode: 3
Extract Data: 01101000 01100101 01101100 01101100 01101111 00100000 01110111 0110
1111 01110010 01101100 01100100
```

```
Trivial File Transfer Protocol
    Opcode: Data Packet (3)
    [Destination File: abc.txt]
    [Read Request in frame 97]
    Block: 1
    [Full Block Number: 1]
  Data (11 bytes)
    Data: 68 65 6c 6c 6f 20 77 6f 72 6c 64
    [Length: 11]
```

```
0000  10 63 c8 5f 57 11 30 9c  23 63 6f c3 08 00 45 00   ·c·_W·0· #co···E·
0010  00 2b 63 39 00 00 80 11  24 3f c0 a8 18 fd c0 a8   ·+c9···· $?·····
0020  18 fc f4 d3 c3 bc 00 17  02 13 00 03 00 01 68 65   ········ ····--he
0030  6c 6c 6f 20 77 6f 72 6c  64 00 00 00              llo worl d···
```

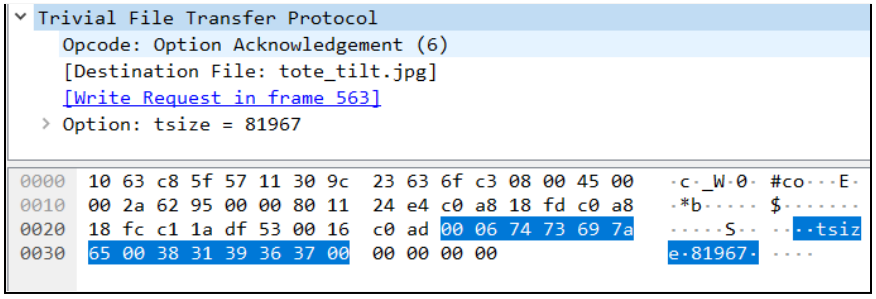
3. ACK Packet

```
ACK Packet
System:
System Hex from Processed Byte: 00040054
System Bits: 00000000 00000100 00000000 01010100
Wireshark:
Wireshark Hex Raw: 00040054
Wireshark Bits: 00000000 00000100 00000000 01010100
isACK: true
extractACK: Block 84
```



4. OACK Packet

```
OACK Packet
System:
System Hex from Processed Byte: 00067473697a6500383139363700
System Bits: 00000000 00000110 01110100 01110011 01101001 01111010 01100101 00000
000 00111000 00110001 00111001 00110110 00110111 00000000
Wireshark:
Wireshark Hex Raw: 00067473697a6500383139363700
Wireshark Bits: 00000000 00000110 01110100 01110011 01101001 01111010 01100101 00
000000 00111000 00110001 00111001 00110110 00110111 00000000
extractOACK: {tsize}, {81967}
```



5. Read Request (With and Without OptsVals)

```
RRQ Packet
System:
System Hex from Processed Byte: 00016e656e656368692e706e67006f637465740074736 97a65003000
System Bits: 00000000 00000001 01101110 01100101 01101110 01100101 01100011 01101000 01101001 00101110 01110000 01101110 01100111 00000000 01101111 01100011 01110100 01100101 01110100 00000000 01110100 01110011 01101001 01111010 01100101 00000000 00110000 00000000
Wireshark:
Wireshark Hex Raw: 00016e656e656368692e706e67006f637465740074736 97a65003000
Wireshark Bits: 00000000 00000001 01101110 01100101 01101110 01100101 01100011 01101000 01101001 00101110 01110000 01101110 01100111 00000000 01101111 01100011 01110100 01100101 01110100 00000000 01110100 01110011 01101001 01111010 01100101 00000000 00110000 00000000
RQHasOACK: true
extractOACKFromRQ: {tsize}, {0}
```

```
RRQ Packet Without Opts & Vals
System:
System Hex from Processed Byte: 00016e656e656368692e706e67006f6374657400
System Bits: 00000000 00000001 01101110 01100101 01101110 01100101 01100011 01101000 01101001 00101110 01110000 01101110 01100111 00000000 01101111 01100011 01110100 01100101 01110100 00000000
Wireshark:
Wireshark Hex Raw: 00016e656e656368692e706e67006f6374657400
RQHasOACK: false
Wireshark Bits: 00000000 00000001 01101110 01100101 01101110 01100101 01100011 01101000 01101001 00101110 01110000 01101110 01100111 00000000 01101111 01100011 01110100 01100101 01110100 00000000
```

```
Trivial File Transfer Protocol
    Opcode: Read Request (1)
    Source File: nenechi.png
    Type: octet
  > Option: tsize = 0

0000  30 9c 23 63 6f c3 10 63  c8 5f 57 11 08 00 45 00   0·#co··c ·_W···E·
0010  00 38 12 e1 00 00 80 11  74 8a c0 a8 18 fc c0 a8   ·8····· t·······
0020  18 fd c4 26 00 45 00 24  3c 67 00 01 6e 65 6e 65   ···&·E·$ <g··nene
0030  63 68 69 2e 70 6e 67 00  6f 63 74 65 74 00 74 73   chi.png· octet·ts
0040  69 7a 65 00 30 00                                 ize·0·
```

6. Write Request (With and Without OptsVals)

```
WRQ Packet
System:
System Hex from Processed Byte: 0002746f74655f74696c742e6a7067006f637465740074736
97a6500383139363700
System Bits: 00000000 00000010 01110100 01101111 01110100 01100101 01011111 01110
100 01101001 01101100 01110100 00101110 01101010 01110000 01100111 00000000 01101
111 01100011 01110100 01100101 01110100 00000000 01110100 01110011 01101001 01111
010 01100101 00000000 00111000 00110001 00111001 00110110 00110111 00000000
Wireshark:
Wireshark Hex Raw: 0002746f74655f74696c742e6a7067006f63746574007473697a6500383139
363700
RQHasOACK: true
extractOACKFromRQ: {tsize}, {81967}
Wireshark Bits: 00000000 00000010 01110100 01101111 01110100 01100101 01011111 01
110100 01101001 01101100 01110100 00101110 01101010 01110000 01100111 00000000 01
101111 01100011 01110100 01100101 01110100 00000000 01110100 01110011 01101001 01
111010 01100101 00000000 00111000 00110001 00111001 00110110 00110111 00000000
```

```
WRQ Packet Without Opts & Vals
System:
System Hex from Processed Byte: 0002746f74655f74696c742e6a7067006f6374657400
System Bits: 00000000 00000010 01110100 01101111 01110100 01100101 01011111 01110
100 01101001 01101100 01110100 00101110 01101010 01110000 01100111 00000000 01101
111 01100011 01110100 01100101 01110100 00000000
Wireshark:
Wireshark Hex Raw: 0002746f74655f74696c742e6a7067006f6374657400
RQHasOACK: false
Wireshark Bits: 00000000 00000010 01110100 01101111 01110100 01100101 01011111 01
110100 01101001 01101100 01110100 00101110 01101010 01110000 01100111 00000000 01
101111 01100011 01110100 01100101 01110100 00000000
```

```
▾ Trivial File Transfer Protocol
    Opcode: Write Request (2)
    Destination File: tote_tilt.jpg
    Type: octet
  › Option: tsize = 81967
```

```
0000  30 9c 23 63 6f c3 10 63  c8 5f 57 11 08 00 45 00   0·#co··c ·_W···E·
0010  00 3e 12 e5 00 00 80 11  74 80 c0 a8 18 fc c0 a8   ·>····· t·······
0020  18 fd df 53 00 45 00 2a  32 41 00 02 74 6f 74 65   ···S·E·* 2A··tote
0030  5f 74 69 6c 74 2e 6a 70  67 00 6f 63 74 65 74 00   _tilt.jp g·octet·
0040  74 73 69 7a 65 00 38 31  39 36 37 00               tsize·81 967·
```

7. Data Packet (wrong output)
   The output is possibly wrong due to issues in decoding in and out of byte[] and Hex in Java. The function for decoding byte to Hex was done through Integer.parseInt(<Str>,16) & Integer.toHexString(<byte>) which has issues with overflow from Hex values of 80 and above as shown in the code below.

```
Data Packet
System:
System Hex from Processed Byte: 0003000ffffffa1ffffff9452ffffffd1409452d140ffffff
9452ffffffd1409452d140ffffff9452ffffffd1409452d140ffffff9452ffffffd1409452d140fff
fff9452ffffffd1409452d141fffffffffffffd9
System Bits: 00000000 00000011 00000000 00001111 11111111 11111111 11111010 00011
111 11111111 11111111 11111001 01000101 00101111 11111111 11111111 11111101 00010
100 00001001 01000101 00101101 00010100 00001111 11111111 11111111 11111001 01000
101 00101111 11111111 11111111 11111101 00010100 00001001 01000101 00101101 00010
100 00001111 11111111 11111111 11111001 01000101 00101111 11111111 11111111 11111
101 00010100 00001001 01000101 00101101 00010100 00001111 11111111 11111111 11111
001 01000101 00101111 11111111 11111111 11111101 00010100 00001001 01000101 00101
101 00010100 00001111 11111111 11111111 11111001 01000101 00101111 11111111 11111
111 11111101 00010100 00001001 01000101 00101101 00010100 00011111 11111111 11111
111 11111111 11111111 11111111 11111111 11111101
Wireshark:
Wireshark Hex Raw: 000300a19452d14009452d14009452d14009452d14009452d14009452d1400
9452d14009452d14009452d14009452d1401ffd9
Wireshark Bits: 00000000 00000011 00000000 10100001 10010100 01010010 11010001 01
000000 00001001 01000101 00101101 00010100 00000000 10010100 01010010 11010001 01
000000 00001001 01000101 00101101 00010100 00000000 10010100 01010010 11010001 01
000000 00001001 01000101 00101101 00010100 00000000 10010100 01010010 11010001 01
000000 00001001 01000101 00101101 00010100 00000000 10010100 01010010 11010001 01
000000 00001001 01000101 00101101 00010100 00000001 11111111 11011001
```

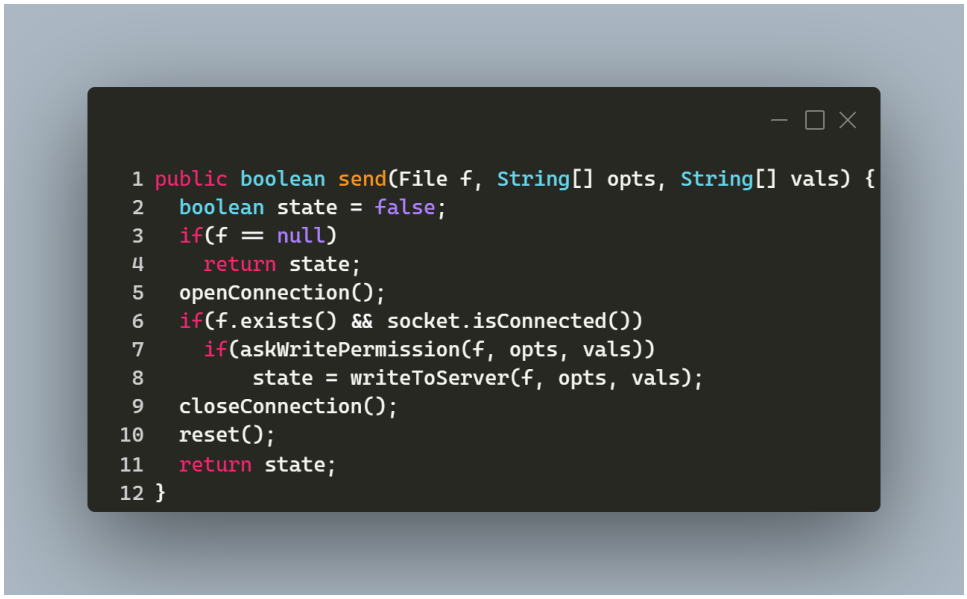
```java
public static void main(String[] args) {
  String src = "0123456789abcdef";
  System.out.println("Raw: byte, hexString");
  for(int i = 0; i < src.length(); i++){
    for(int j = 0; j < src.length(); j++){
      String raw = src.charAt(i) + "" +src.charAt(j);
      Integer hex = Integer.parseInt(raw,16); //String to Hex's Integer Equivalent
      byte b = hex.byteValue(); //Byte value of Hex
      String hexString = Integer.toHexString(b); //Hex string of Hex's byte value
      System.out.println(raw + ": " + b + ", " + hexString);

      //SAME THING GOES IF hex IS IN SHORT DATA TYPE.
    }
  }
}
```

```
78: 120, 78
79: 121, 79
7a: 122, 7a
7b: 123, 7b
7c: 124, 7c
7d: 125, 7d
7e: 126, 7e
7f: 127, 7f
80: -128, ffffff80
81: -127, ffffff81
82: -126, ffffff82
83: -125, ffffff83
84: -124, ffffff84
85: -123, ffffff85
86: -122, ffffff86
87: -121, ffffff87
88: -120, ffffff88
```

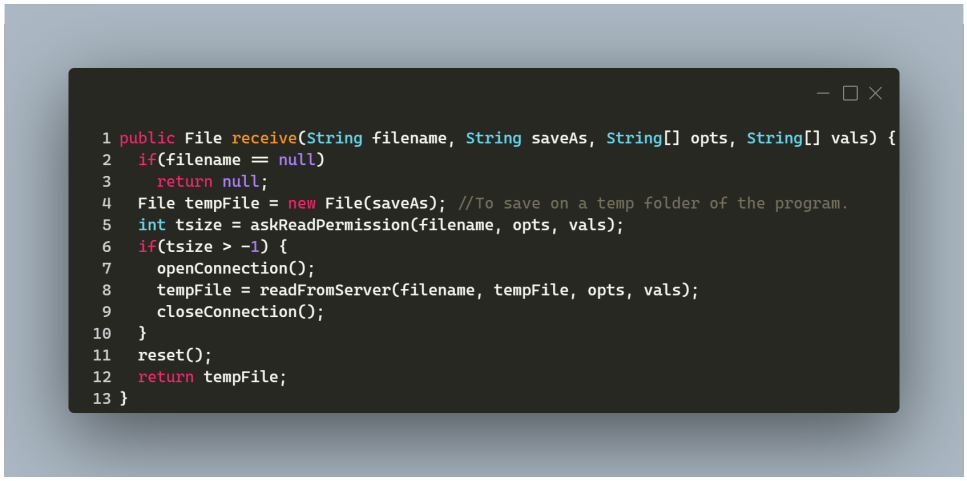
Network Sequence Code

The function call for sending and receiving is abstracted publicly into two functions which delegate the entire network related processes of the TFTP connection prior to the actual transmission.

The sequences contain the opening and closing of the socket connection as it is based on the understanding that TFTP only requires a connection if the user has an intent of conducting a TFTP transmission. It also handles the permission calls and assessment as well as the call for reading or writing to a server if permission is granted. The connection configuration is specified from the function's object instantiator.



```java
public boolean send(File f, String[] opts, String[] vals) {
  boolean state = false;
  if(f == null)
    return state;
  openConnection();
  if(f.exists() && socket.isConnected())
    if(askWritePermission(f, opts, vals))
        state = writeToServer(f, opts, vals);
  closeConnection();
  reset();
  return state;
}
```



```java
public File receive(String filename, String saveAs, String[] opts, String[] vals) {
  if(filename == null)
    return null;
  File tempFile = new File(saveAs); //To save on a temp folder of the program.
  int tsize = askReadPermission(filename, opts, vals);
  if(tsize > -1) {
    openConnection();
    tempFile = readFromServer(filename, tempFile, opts, vals);
    closeConnection();
  }
  reset();
  return tempFile;
}
```

TFTP Sequence Diagrams and Code

For the sequence diagram and code, the following already follows the code mentioned in the Network Sequence which calls for askWritePermission() and askReadPermission() as well as the writeToServer() and readFromServer(). The sequence that follows shows the process diagram and equivalent pseudocode.

[SEQUENCE]

```
readFromServer(){
    OutputStream outstream = new FileOutputStream(file);
    boolean isTerminating = false;
    BUFFERSIZE = 512; //Depending on the blocksize agreed upon.
    do {
        //Receive TFTP packet from server
        byte[] recv = new byte[this.BUFFER_SIZE];
        packet = new DatagramPacket(recv, recv.length);
        socket.receive(packet);

        //Determine if is an Error Packet
        if(tftp.isError(packet)){
            //Determine if error is Error Codes 1 to 3
            String[] err = tftp.extractError(packet);
            if(err[0].equalsIgnoreCase("1")) {
                //Pop-up Message: File not found
            }else if(err[0].equalsIgnoreCase("2")){
                //Pop-up Message: File not found
            }else if(err[0].equalsIgnoreCase("3")){
                //Pop-up Message: File not found
            }else{
                //Pop-up Message: Fatal error occurred.
            }
            return false;
        }else{ //Assume as Data Packet
            //Extract contents to buffer
            buffer = tftp.extractData(packet);

            //Extract and send Block# as ACK
            int ctr = tftp.extractBlockNumber(packet);
            if(ctr > -1)
                socket.send(tftp.getACK(ctr));

            //Write buffer to file
            int bytesRead = packet.getLength(); //BYTE LENGTH OF PACKET'S DATA SEGMENT; DETERMINES THE SMALLEST BYTE DATAGRAM RECEIVED, (I.E. ELIMINATING 0s REMAINING IF PACKET IS < RECV)
            if(bytesRead < BUFFER_SIZE)
                isTerminating = true;
            outputStream.write(buffer, 0, bytesRead); //WRITES THE DATA BUFFER TO FILE
        }
    }while(!isTerminating);
    return true;
}
```

```
writeToServer(){
    //Prepare files

    //Preparatory for reading opts and vals

    //Block Counter
    int ctr = 1; //0th block already at request.
    BUFFER_SIZE = 512; //Depending on the blocksize agreed upon.
    while((bytesRead = inputStream.read(buffer)) != -1) { //While file not done streaming.
        //Build TFTP Data Packet
        byte[] p = tftp.getDataPacket(ctr, buffer);

        //Send TFTP Packet
        socket.send(p);

        //Await and Receive ACK or Error Packet
        do{
            boolean valid = false;

            byte[] recv = new byte[this.BUFFER_SIZE];
            packet = new DatagramPacket(recv, recv.length);
            socket.receive(packet);

            if(tftp.isError(packet)){
                //Determine if error is Error Codes 1 to 3
                String[] err = tftp.extractError(packet);
                if(err[0].equalsIgnoreCase("1")) {
                    //Pop-up Message: File not found
                }else if(err[0].equalsIgnoreCase("2")){
                    //Pop-up Message: File not found
                }else if(err[0].equalsIgnoreCase("3")){
                    //Pop-up Message: File not found
                }else{
                    //Pop-up Message: Fatal error occurred.
                }
                return null;
            }else{
                //Read ACK and retrieve Block#
                int blockNum = tftp.extractBlockNumber(packet);

                //Check if blockNum is equal to what was sent
                if(blockNum > 0){
                    if(blockNum == ctr)
                        valid = true;
                }else{
                    //Keep asking for packets until a valid (equal Block#) is received.
                }
            }
        }while(!valid);


        //DO NOT MOVE THIS. LET IT BE PLACED LAST.
        if(inputStream.available() < BUFFER_SIZE) {
            BUFFER_SIZE = inputStream.available();
            buffer = new byte[BUFFER_SIZE];
        }
    }
}
```

```
//Ask Read Permission
askReadPermission(String filename, String[] opts, String[] vals){
    //Reset packet
    packet = null;

    //Build RRQ byte[] and packet.
    byte[] rrq = (filename, opts, vals);
    packet = new DatagramPacket(rrq, rrq.length);


    //Send Request
    socket.send(rrq);

    //Await for OACK or ERROR to arrive.
    byte[] recv = new byte[65535]; //TENTATIVE LENGTH
    packet = new packet(recv, recv.length);
    socket.receive(packet);

    //Read contents
    if(tftp.isError(packet.getBytes())){
        //Received packet is an ERROR
        gui.popError(tftp.extractError(packet.getBytes)); //Display pop message for GUI error
        return -1;
    }else{
        //Received packet is an OACK (already assumes as OACK)
        if(confirmEqual(wrq, packet.getBytes()))
            return getTvals(packet.getBytes()); //Returns the Tval value which indicates the filesize of the file. Return 0 if not found.
        return -1;
    }
}
```

```
//Ask Write Permission
askWritePermission(File f, String[] opts, String[] vals){
    //Reset packet
    packet = null;

    //Build WRQ byte[] and packet.
    byte[] wrq = (f, 'octet', opts, vals);
    packet = new DatagramPacket(wrq, wrq.length); //Declared globally for Client.java

    //Send Request
    socket.send(wrq);

    //Await for OACK or ERROR to arrive.
    byte[] recv = new byte[65535]; //TENTATIVE LENGTH
    packet = new packet(recv, recv.length);
    socket.receive(packet);

    //Read contents
    if(tftp.isError(packet.getBytes())){
        //Received packet is an ERROR
        gui.popError(tftp.extractError(packet.getBytes)); //Display pop message for GUI error
        return false;
    }else{
        //Received packet is an OACK (already assumes as OACK)
        if(confirmEqual(wrq, packet.getBytes())) //Compare OACK if equal
            return true;
        return false;
    }
}

```

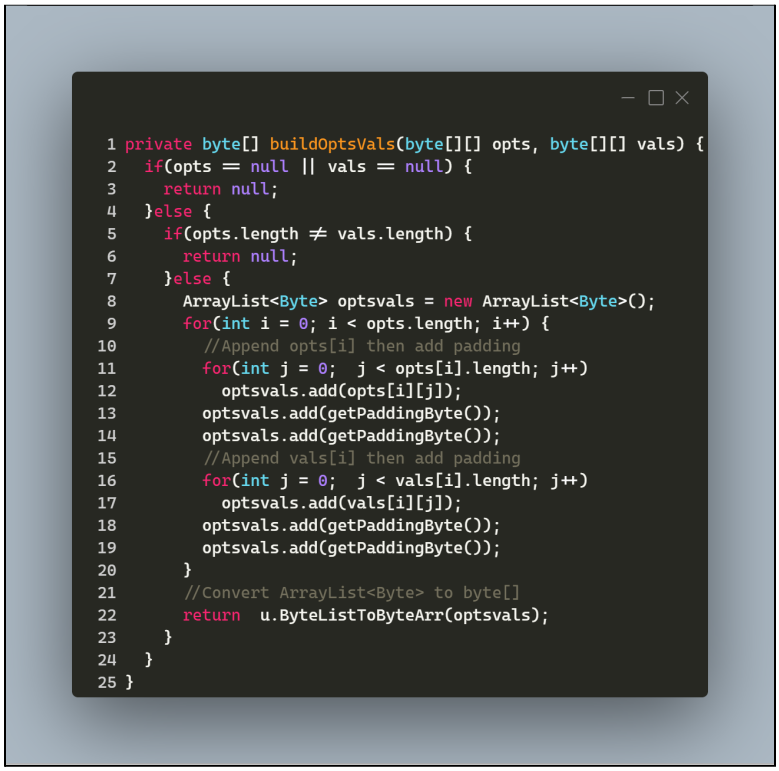
Appendix

1. buildOpcode()

```
private byte[] buildOpcode(byte opcode) {
  byte[] opcodeByte = {getPaddingByte(), getPaddingByte(), getPaddingByte(), opcode};
  return opcodeByte;
}
```

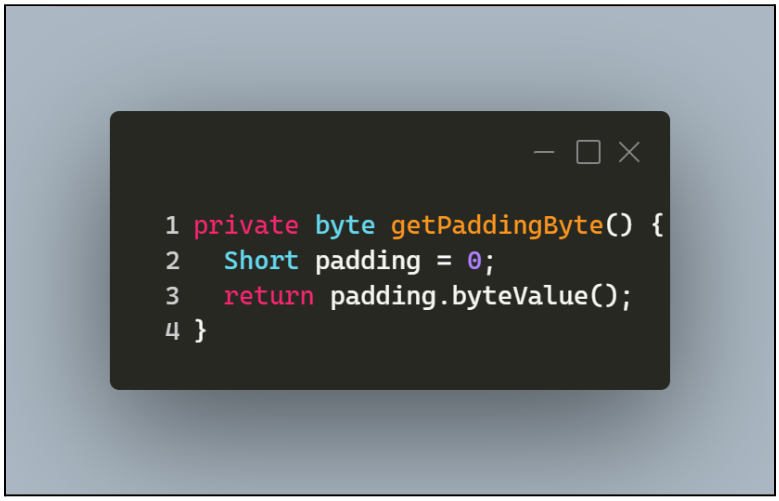
2. buildOptVals()

```
private byte[] buildOptsVals(String[] opts, String[] vals) {
  if(opts == null || vals == null) {
    return null;
  }else {
    if(opts.length != vals.length) {
      return null;
    }else {
      byte[][] optsBytes = stringArrToByteArr(opts);
      byte[][] valsBytes = stringArrToByteArr(vals);
      return buildOptsVals(optsBytes, valsBytes);
    }
  }
}
```



```
private byte[] buildOptsVals(byte[][] opts, byte[][] vals) {
  if(opts == null || vals == null) {
    return null;
  }else {
    if(opts.length != vals.length) {
      return null;
    }else {
      ArrayList<Byte> optsvals = new ArrayList<Byte>();
      for(int i = 0; i < opts.length; i++) {
        //Append opts[i] then add padding
        for(int j = 0;  j < opts[i].length; j++)
          optsvals.add(opts[i][j]);
        optsvals.add(getPaddingByte());
        optsvals.add(getPaddingByte());
        //Append vals[i] then add padding
        for(int j = 0;  j < vals[i].length; j++)
          optsvals.add(vals[i][j]);
        optsvals.add(getPaddingByte());
        optsvals.add(getPaddingByte());
      }
      //Convert ArrayList<Byte> to byte[]
      return  u.ByteListToByteArr(optsvals);
    }
  }
}
```

3. getPaddingByte()



```
1 private byte getPaddingByte() {
2   Short padding = 0;
3   return padding.byteValue();
4 }
```

4. getPaddingByteArr()

```
1 private byte[] getPaddingByteArr() {
2   byte[] arr = {getPaddingByte()};
3   return arr;
4 }
```

5. combineBytes()

```
1 private byte[] combineBytes(byte[][] bytes){
2   int size = 0, ctr = 0;
3   for(int i = 0; i < bytes.length; i++)
4     size += bytes[i].length;
5   byte[] combinedBytes = new byte[size];
6   for(byte[] byteArr: bytes) {
7     for(byte b: byteArr) {
8         combinedBytes[ctr] = b;
9         ctr++;
10    }
11  }
12  return combinedBytes;
13 }
```

6. GUI Layout

Some of the GUI's components were referenced from the TFTPd TFTP Client.