NSCOM01: TFTP Client MP

Balcueva, Escalona, Fadrigo, Fortiz

Project Rationale

To create a TFTP Client program that complies with **RFC documents: 1350, 2347, 2358, 2349***

Program Specifications

1. Programming Language: **Java**
2. Interface: **GUI** (with verbose logging using -V)

Features Implemented

- TFTP protocol-based features
  - **Uploading and downloading files** (at unlimited file sizes)
  - **Error detection** and handling
  - **Blocksize modification** (~40Mbps)
  - **Options recognition and compliance** (blksize and tsize only)
- Non-TFTP protocol-based features
  - **Network-based timeouts** (3 seconds)
  - **Network verification before transmission**

Limitations

1. **Does not use the official TFTP timeout option**
2. **Does not verify ACKs** (for both send and receive) due to byte to int conversion limitations*.

* Nature of byte being limited to 0-255 unsigned, thus any ACK values >255 are not recognized properly.

TFTP Packet Assembly: According to RFC 1350

- REQUEST (READ/WRITE, W/O OPTVALS)

Request (w/o OptVals)						
Length	2Bytes		Length of String	1Byte	Length of String	1Byte
Segment	Padding	Type (1/2)	Filename	Padding	Mode	Padding

- DATA

Data				
Length	2Bytes		2Bytes	n Bytes
Segment	Padding	3	Block#	Data

- REQUEST (READ/WRITE, W OPTVALS)

Request (w/o OptVals)										
Length	2Bytes		Length of String	1Byte	Length of String	1Byte	Length of String	Length of String	…	Length of String	Length of String
Segment	Padding	Type (1/2)	Filename	Padding	Mode	Padding	Opt1 (ends in \0)	Val1 (ends in \0)	…	OptN (ends in \0)	ValN (ends in \0)

- ACK

ACK			
Length	2Bytes		2Bytes
Segment	Padding	4	Block#

TFTP Packet Assembly: According to RFC 1350

- OACK (Optional ACK)

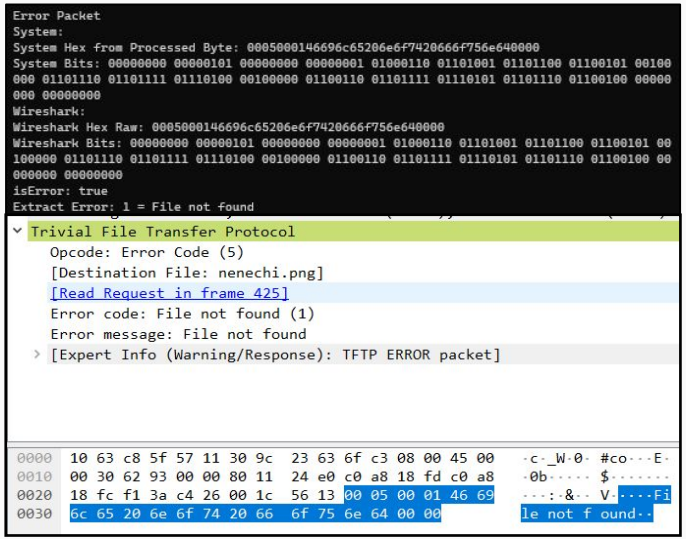
OACK							
Length	2Bytes		Length of String	Length of String	…	Length of String	Length of String
Segment	Padding	6	Opt1 (ends in \0)	Val1 (ends in \0)	…	OptN (ends in \0)	ValN (ends in \0)

- ERROR PACKET

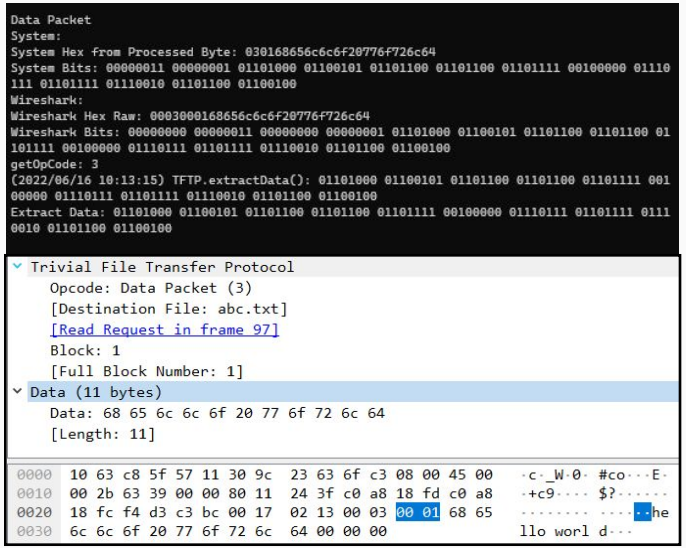
Error					
Length	2Bytes		2Bytes	Length of String	1Byte
Segment	Padding	5	Error Code	ErrMsg	Padding

TFTP Packet Assembly: Results and Reference

Error Packet



Data Packet



TFTP Packet Assembly: Results and Reference

ACK Packet

```
ACK Packet
System:
System Hex from Processed Byte: 04054
System Bits: 00000100 00000101
Wireshark:
Wireshark Hex Raw: 00040054
Wireshark Bits: 00000000 00000100 00000000 01010100
isACK: true
extractACK: Block 84
```

```
Trivial File Transfer Protocol
    Opcode: Acknowledgement (4)
    [Destination File: tote_tilt.jpg]
    [Write Request in frame 563]
    Block: 84
    [Full Block Number: 84]
```

```
0000  10 63 c8 5f 57 11 30 9c  23 63 6f c3 08 00 45 00   ·c·_W·0· #co···E·
0010  00 20 62 e9 00 00 80 11  24 9a c0 a8 18 fd c0 a8   · b····· $·······
0020  18 fc c1 1a df 53 00 0c  ab c5 00 04 00 54 00 00   ·····S·· ·····T··
0030  00 00 00 00 00 00 00 00  00 00 00 00              ···· ····
```

OACK Packet

```
OACK Packet
System:
System Hex from Processed Byte: 067473697a65038313936370
System Bits: 00000110 01110100 01110011 01101001 01111010 01100101 00000011 10000011 00010
011 10010011 01100011 01110000
Wireshark:
Wireshark Hex Raw: 00067473697a6500383139363700
Wireshark Bits: 00000000 00000110 01110100 01110011 01101001 01111010 01100101 00000000 00
111000 00110001 00111001 00110110 00110111 00000000
extractOACK: {tsize}, {81967}
```

```
Trivial File Transfer Protocol
    Opcode: Option Acknowledgement (6)
    [Destination File: tote_tilt.jpg]
    [Write Request in frame 563]
  › Option: tsize = 81967
```

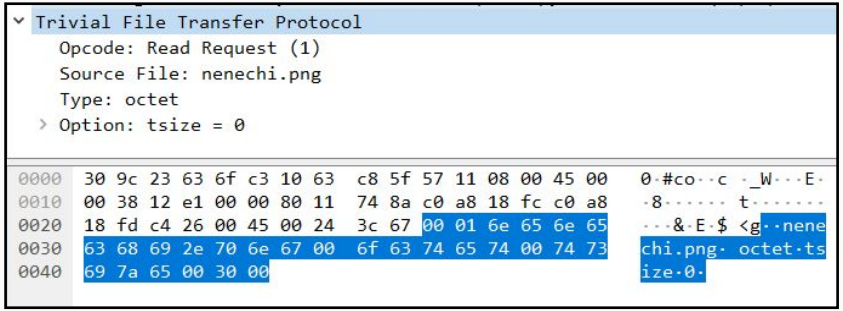
```
0000  10 63 c8 5f 57 11 30 9c  23 63 6f c3 08 00 45 00   ·c·_W·0· #co···E·
0010  00 2a 62 95 00 00 80 11  24 e4 c0 a8 18 fd c0 a8   ·*b····· $·······
0020  18 fc c1 1a df 53 00 16  c0 ad 00 06 74 73 69 7a   ·····S·· ····tsiz
0030  65 00 38 31 39 36 37 00  00 00 00 00              e·81967· ····
```

TFTP Packet Assembly: Results and Reference

Read Request (With and Without OptsVals)

```
RRQ Packet
System:
System Hex from Processed Byte: 016e656e656368692e706e6706f6374657407473697a650300
System Bits: 00000001 01101110 01100101 01101110 01100101 01100011 01101000 01101001 00101
110 01110000 01101110 01100111 00000110 11110110 00110111 01000110 01010111 01000000 01110
100 01110011 01101001 01111010 01100101 00000011 00000000
Wireshark:
Wireshark Hex Raw: 00016e656e656368692e706e67006f6374657407473697a65003000
Wireshark Bits: 00000000 00000001 01101110 01100101 01101110 01100101 01100011 01101000 01
101001 00101110 01110000 01101110 01100111 00000000 01101111 01100011 01110100 01100101 01
110100 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: 016e656e656368692e706e6706f637465740
System Bits: 00000001 01101110 01100101 01101110 01100101 01100011 01101000 01101001 00101
110 01110000 01101110 01100111 00000110 11110110 00110111 01000110 01010111 01000000
Wireshark:
Wireshark Hex Raw: 00016e656e656368692e706e67006f6374657400
RQHasOACK: false
Wireshark Bits: 00000000 00000001 01101110 01100101 01101110 01100101 01100011 01101000 01
101001 00101110 01110000 01101110 01100111 00000000 01101111 01100011 01110100 01100101 01
110100 00000000
```



TFTP Packet Assembly: Results and Reference

Write Request (With and Without OptsVals)

```
WRQ Packet
System:
System Hex from Processed Byte: 02746573742e706e6706f63746574074736 97a65038313936370
System Bits: 00000010 01110100 01100101 01110011 01110100 00101110 01110000 01101110 01100
111 00000110 11110110 00110111 01000110 01010111 01000000 01110100 01110011 01101001 01111
010 01100101 00000011 10000011 00010011 10010011 01100011 01110000
Wireshark:
Wireshark Hex Raw: 0002746f74655f74696c742e6a7067006f63746574007473697a6500383139363700
RQHasOACK: true
extractOACKFromRQ: {tsize}, {81967}
Wireshark Bits: 00000000 00000010 01110100 01101111 01110100 01100101 01011111 01110100 01
101001 01101100 01110100 00101110 01101010 01110000 01100111 00000000 01101111 01100011 01
110100 01100101 01110100 00000000 01110100 01110011 01101001 01111010 01100101 00000000 00
111000 00110001 00111001 00110110 00110111 00000000
```

```
WRQ Packet Without Opts & Vals
System:
System Hex from Processed Byte: 02746573742e706e6706f637465740
System Bits: 00000010 01110100 01100101 01110011 01110100 00101110 01110000 01101110 01100
111 00000110 11110110 00110111 01000110 01010111 01000000
Wireshark:
Wireshark Hex Raw: 0002746f74655f74696c742e6a7067006f6374657400
RQHasOACK: false
Wireshark Bits: 00000000 00000010 01110100 01101111 01110100 01100101 01011111 01110100 01
101001 01101100 01110100 00101110 01101010 01110000 01100111 00000000 01101111 01100011 01
110100 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·
```

Network Sequence

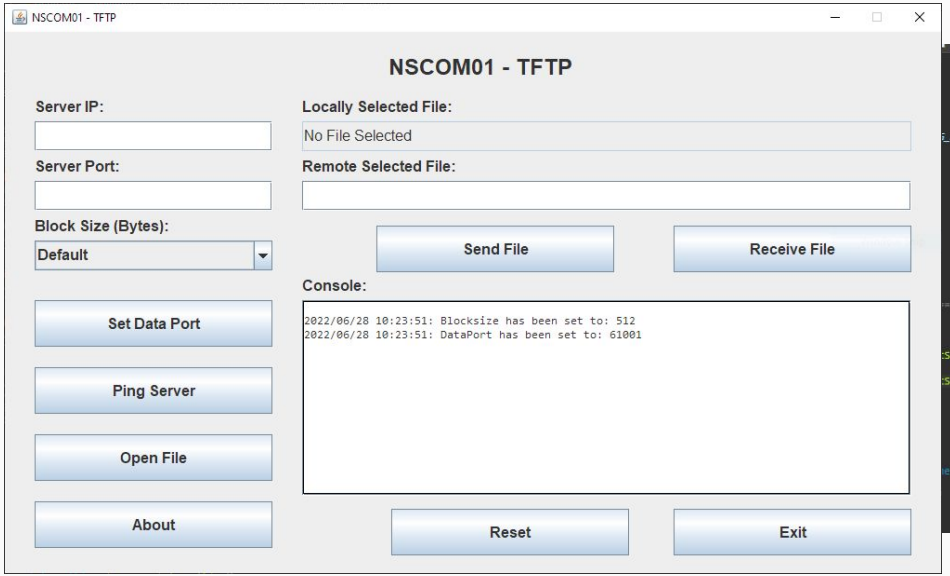
Send

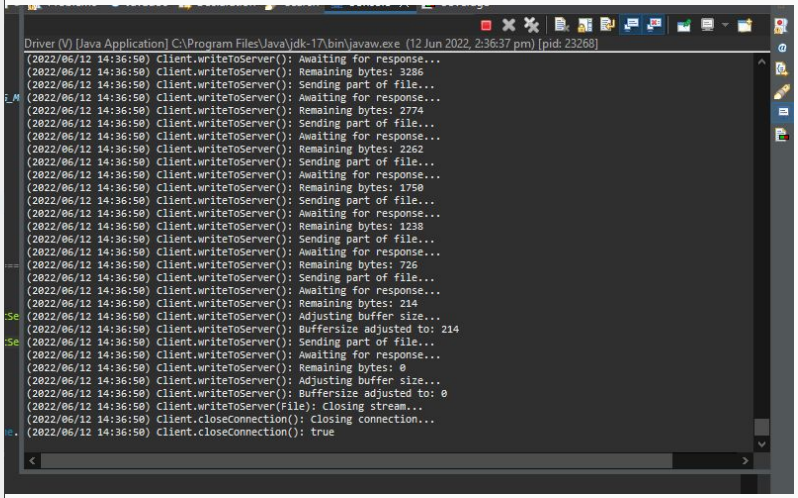
```java
public boolean send(File f, String[] opts, String[] vals) {
  boolean state = false;
  if(f == null)
    return state;
  openConnection();
  if(f.exists() && socket.isConnected())
    if(askWritePermission(f, opts, vals))
        state = writeToServer(f, opts, vals);
  closeConnection();
  reset();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

Receive

```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;
}
```

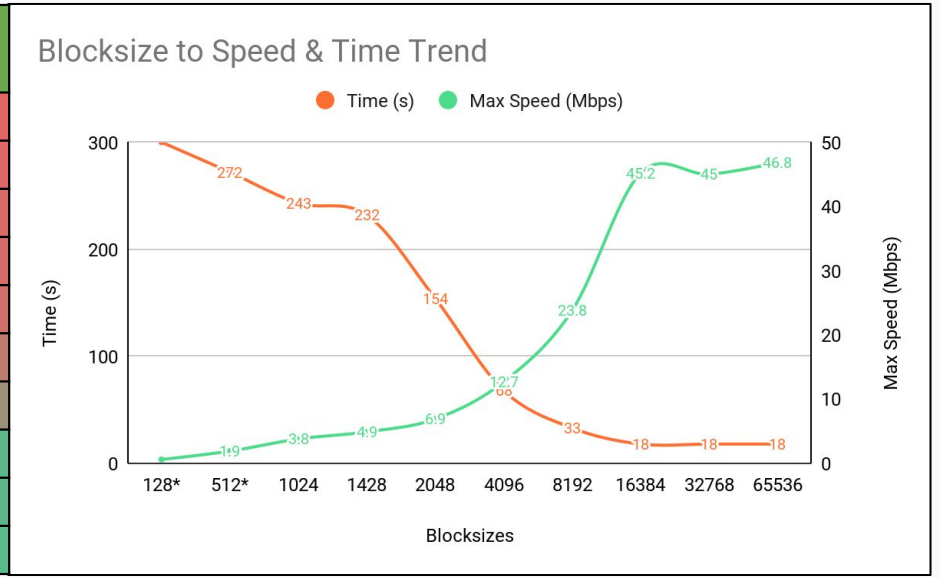
GUI Layout (+ Verbose Mode)





Benchmark

Blocksize (bytes)	Time (s)	Max Speed (Mbps)
128*	300	0.6
512*	272	1.9
1024	243	3.8
1428	232	4.9
2048	154	6.9
4096	68	12.7
8192	33	23.8
16384	18	45.2
32768	18	45
65536	18	46.8



The benchmark file is a video file with a size of 88MB

Git Repository Link

https://github.com/jm55DLSU/NSCOM01

*Do note that the project was developed through Eclipse or IntelliJ IDEs

みんな私たちに聞いてくれありがとうございました!

まもなくデモーをします