**Team 2: Final Project**

Kenan El-Gaouny

Cyrus Sadeghi

Eliab Woldeyes

Majeed Mirza

Rishabh Singh

Table of Contents

[Responsibilities for each Iteration: 3](#_Toc452933903)

[Iteration #1 3](#_Toc452933904)

[Iteration #2 3](#_Toc452933905)

[Iteration #3 4](#_Toc452933906)

[Iteration #4 5](#_Toc452933907)

[Iteration #5 6](#_Toc452933908)

[UCM Diagrams: 6](#_Toc452933909)

[Timing Diagrams: 11](#_Toc452933910)

[Normal Operation: 11](#_Toc452933911)

[Delayed Packets: 11](#_Toc452933912)

[Lost and Duplicate packets: 12](#_Toc452933913)

[Invalid TID (05): 13](#_Toc452933914)

[Invalid Request (04): 14](#_Toc452933915)

[Invalid Data (04): 14](#_Toc452933916)

[File Not Found (01): 15](#_Toc452933917)

[Access Violation (02) 15](#_Toc452933918)

[Disk Full (03) 17](#_Toc452933919)

[File Already Exists (06) 18](#_Toc452933920)

[UML Class Diagrams: 19](#_Toc452933921)

[Instructions: 21](#_Toc452933922)

[Setup Instructions 21](#_Toc452933923)

[Testing Instructions 21](#_Toc452933924)

[Test Cases 23](#_Toc452933925)

[Additional tests performed: 23](#_Toc452933926)

[Iteration #4 23](#_Toc452933927)

[Iteration #5 27](#_Toc452933928)

# Responsibilities for each Iteration:

## Iteration #1

Cyrus Sadeghi 100934822

* Client implementation and testing
  + Implemented TFTP reading/writing
* TFTPServer class implementation and testing
  + Implemented spawning multiple ClientConnectionThreads
* Bug fixes
* Created FileOperation class
  + Used for splitting files into data blocks and merging blocks to original file
* Created Github Repository
* General testing
* README.txt

Eliab Woldeyes 100937656

* ClientConnectionThread implementation and testing
  + Implemented TFTP reading/writing and error checking
* Server implementation and testing
* Bug fixes
* General testing

Kenan El-Gaouny 100935277

* ClientConnectionThread implementation and testing
  + Implemented TFTP reading/writing and error checking
* Intermediate Host changes for TFTP packet transferring
* Bug fixes
* General testing

Majeed Mirza 100943050

* UCM Diagrams
* Testing writeup
* Documentation writeup
* General testing

Rishabh Singh 100941951

* UML Diagrams
* General testing

## Iteration #2

Cyrus Sadeghi 100934822

* Refactored majority of code
* Made TFTPIntHost multithreaded
* Major bug fixes in all classes
* General testing
* README.txt

Eliab Woldeyes 100937656

* Created TFTPCommon class
* Cleaned code
* Timing Diagrams
* Bug fixes
* General testing

Kenan El-Gaouny 100935277

* Implemented network error handling in TFTPServer
* Major bug fixes in all classes
* General testing

Majeed Mirza 100943050

* TFTPIntHost error simulation cases
* Responsibilities write-up
* Documentation
* Bug fixes
* General testing

Rishabh Singh 100941951

* UML Diagrams
* Implemented network error handling in TFTPClient
* Bug fixes
* General testing

## Iteration #3

Cyrus Sadeghi 100934822

* Built in depth UI for Error Simulator
* Error Simulator dealing with error packets
* Client and server error packet details
* Bug fixes
* General testing
* README.txt

Eliab Woldeyes 100937656

* Documentation
* Responsibilities write-up
* In depth testing

Kenan El-Gaouny 100935277

* Timing diagrams
* UML Diagrams
* In depth testing

Majeed Mirza 100943050

* Client and Server sending error packets and handling errors
* Responsibilities write-up
* Bug fixes
* General testing

Rishabh Singh 100941951

* Timing diagrams
* Client and Server receiving error packets and handling errors
* Bug fixes
* General testing

## Iteration #4

Cyrus Sadeghi 100934822

* Updated error handling for I/O errors
* Updated Error Simulator for dealing with I/O error packets
* Added rolling over block numbers for server and client in order to transfer > 65535 packets per file
* Fix problems with Iteration 3
* Client and server error packet details
* Bug fixes
* General testing
* README.txt

Eliab Woldeyes 100937656

* Documentation
* Responsibilities write-up
* General testing

Kenan El-Gaouny 100935277

* Updated error handling for I/O errors
* Bug fixes
* General testing

Majeed Mirza 100943050

* Timing Diagrams
* Responsibilities write-up
* General testing

Rishabh Singh 100941951

* Client and Server receiving error packets and handling errors
* Testing opcode errors and mode errors
* Bug fixes
* General testing

## Iteration #5

Cyrus Sadeghi 100934822

* Implemented multiple computer support
* Changed Error Simulator to run on client computer or server computer
* Fix problems with Iteration 4
* Iteration 5 testing and regression testing
* Diagram and documentation review
* README.txt

Eliab Woldeyes 100937656

* Documentation
* Responsibilities write-up
* General testing

Kenan El-Gaouny 100935277

* Final Documentation and Diagrams
* Testing
* Bug Fixes

Majeed Mirza 100943050

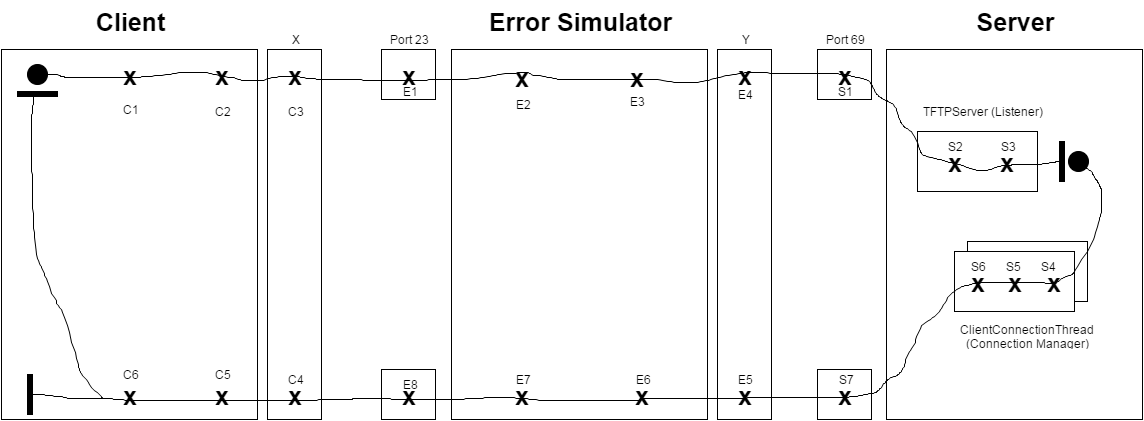
* Testing
* Diagram and documentation review

Rishabh Singh 100941951

* Testing

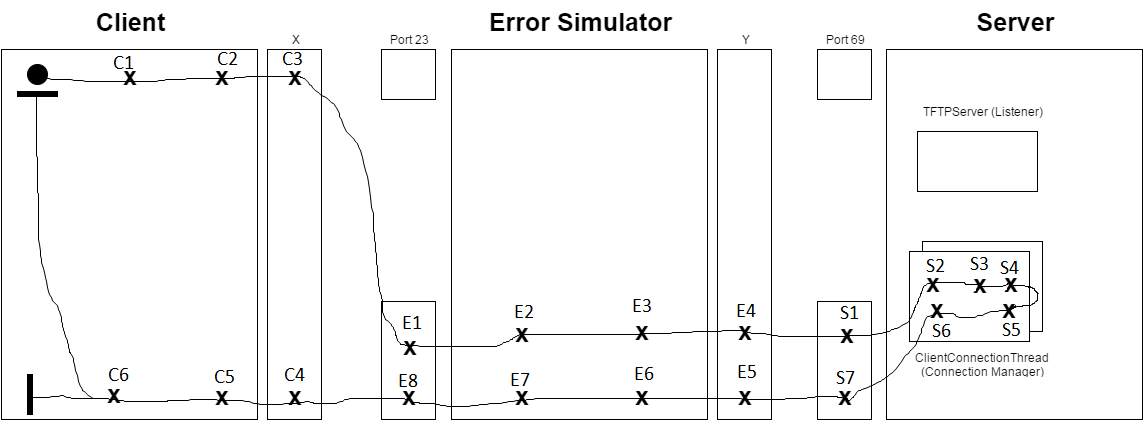
# UCM Diagrams:

**Establish WRQ UCM**



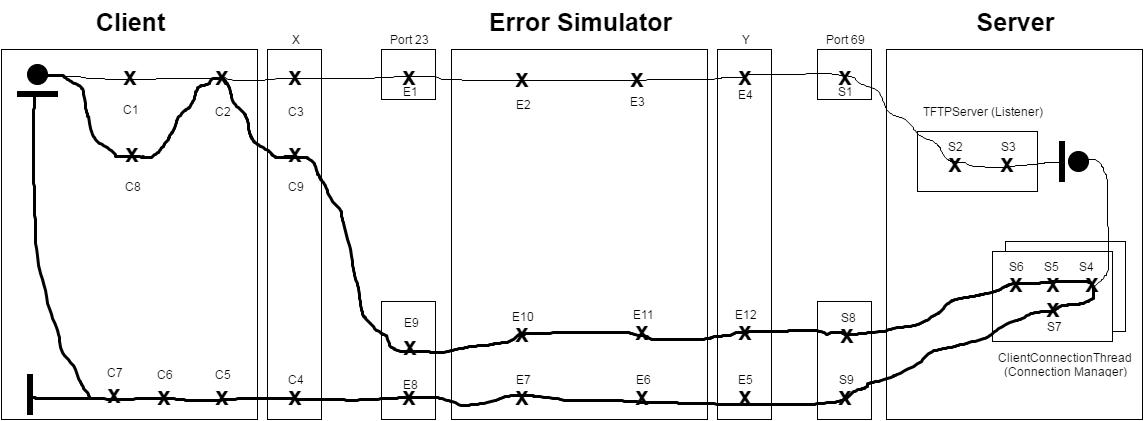
|  |  |  |
| --- | --- | --- |
| C1 – form DATA | E1 – receive datagram | S1 – receive datagram |
| C2 – create datagram | E2 – extract message | S2 – extract DATA |
| C3 – send datagram | E3 – create datagram | S3 – verify WRQ |
| C4 – receive datagram | E4 – send datagram | S4 – create socket |
| C5 – extract message | E5 – receive datagram | S5 – form ACK |
| C6 – verify ACK | E6 – extract message | S6 – create datagram |
|  | E7 – create datagram | S7 – send datagram |
|  | E8 – send datagram |  |

**File Transfer WRQ UCM**



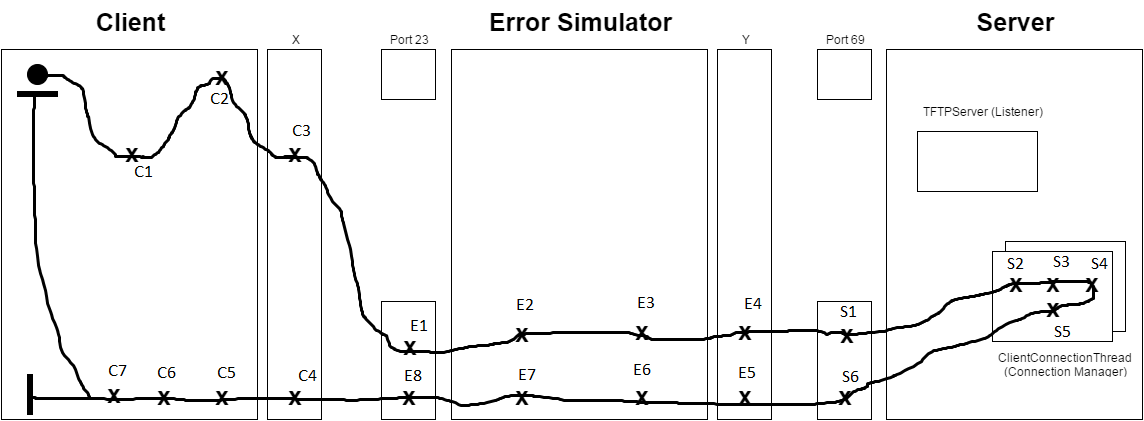
|  |  |  |
| --- | --- | --- |
| C1 – form DATA | E1 – receive datagram | S1 – receive datagram |
| C2 – create datagram | E2 – extract message | S2 – extract DATA |
| C3 – send datagram | E3 – create datagram | S3 – verify DATA |
| C4 – receive datagram | E4 – send datagram | S4 – Write File |
| C5 – extract message | E5 – receive datagram | S5 – form ACK |
| C6 – verify ACK | E6 – extract message | S6 – create datagram |
|  | E7 – create datagram | S7 – send datagram |
|  | E8 – send datagram |  |

**Establish RRQ UCM**



|  |  |  |
| --- | --- | --- |
| C1 – form message | E1 – receive datagram | S1 – receive datagram |
| C2 – create datagram | E2 – extract message | S2 – extract message |
| C3 – send datagram | E3 – create datagram | S3 – verify RRQ |
| C4 – receive datagram | E4 – send datagram | S4 – read DATA |
| C5 – extract DATA | E5 – receive datagram | S5 – verify ACK |
| C6 – verify DATA | E6 – extract message | S6 – extract message |
| C7 – write DATA | E7 – create datagram | S7 – create datagram |
| C8 – form ACK | E8 – send datagram | S8 – receive datagram |
| C9 – send datagram | E9 – receive datagram | S9 – send datagram |
|  | E10 – extract message |  |
|  | E11 – create datagram |  |
|  | E12 – send datagram |  |

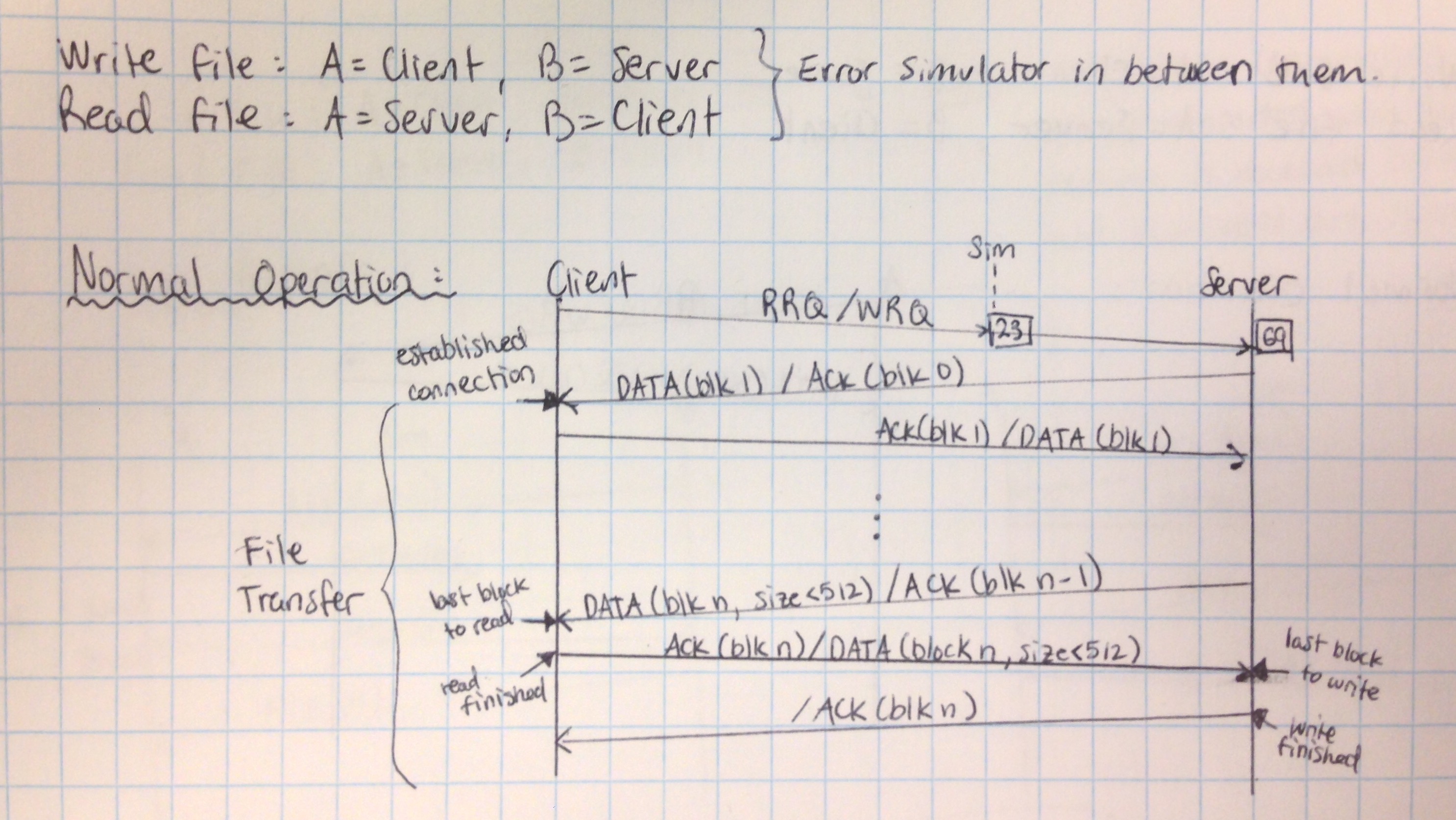
**File Transfer RRQ UCM**



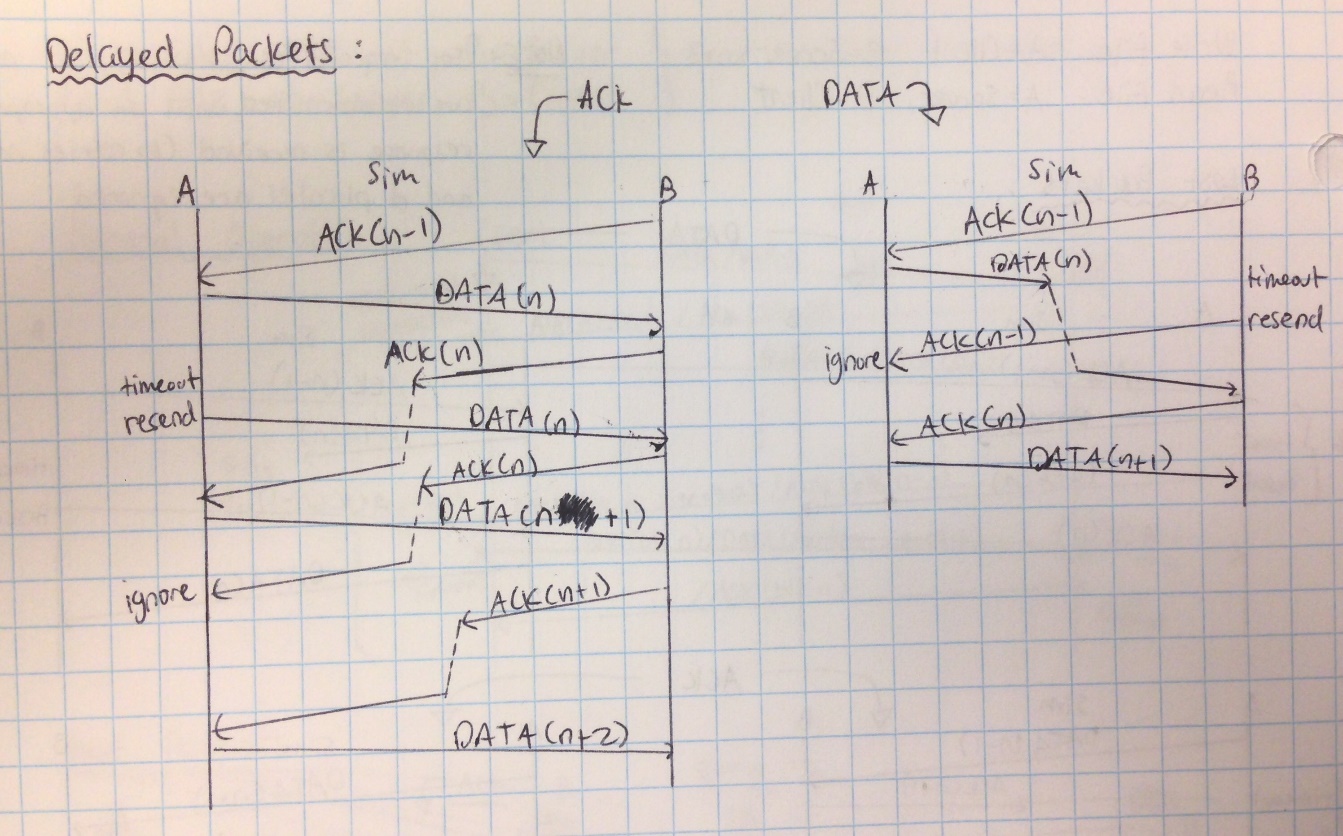
|  |  |  |
| --- | --- | --- |
| C1 – form message | E1 – receive datagram | S1 – receive datagram |
| C2 – create datagram | E2 – extract message | S2 – extract message |
| C3 – send datagram | E3 – create datagram | S3 – verify RRQ |
| C4 – receive datagram | E4 – send datagram | S4 – read DATA |
| C5 – extract DATA | E5 – receive datagram | S5 – form DATA |
| C6 – verify DATA | E6 – extract message | S6 – create datagram |
| C7 – Write DATA | E7 – create datagram |  |
|  | E8 – send datagram |  |

# Timing Diagrams:

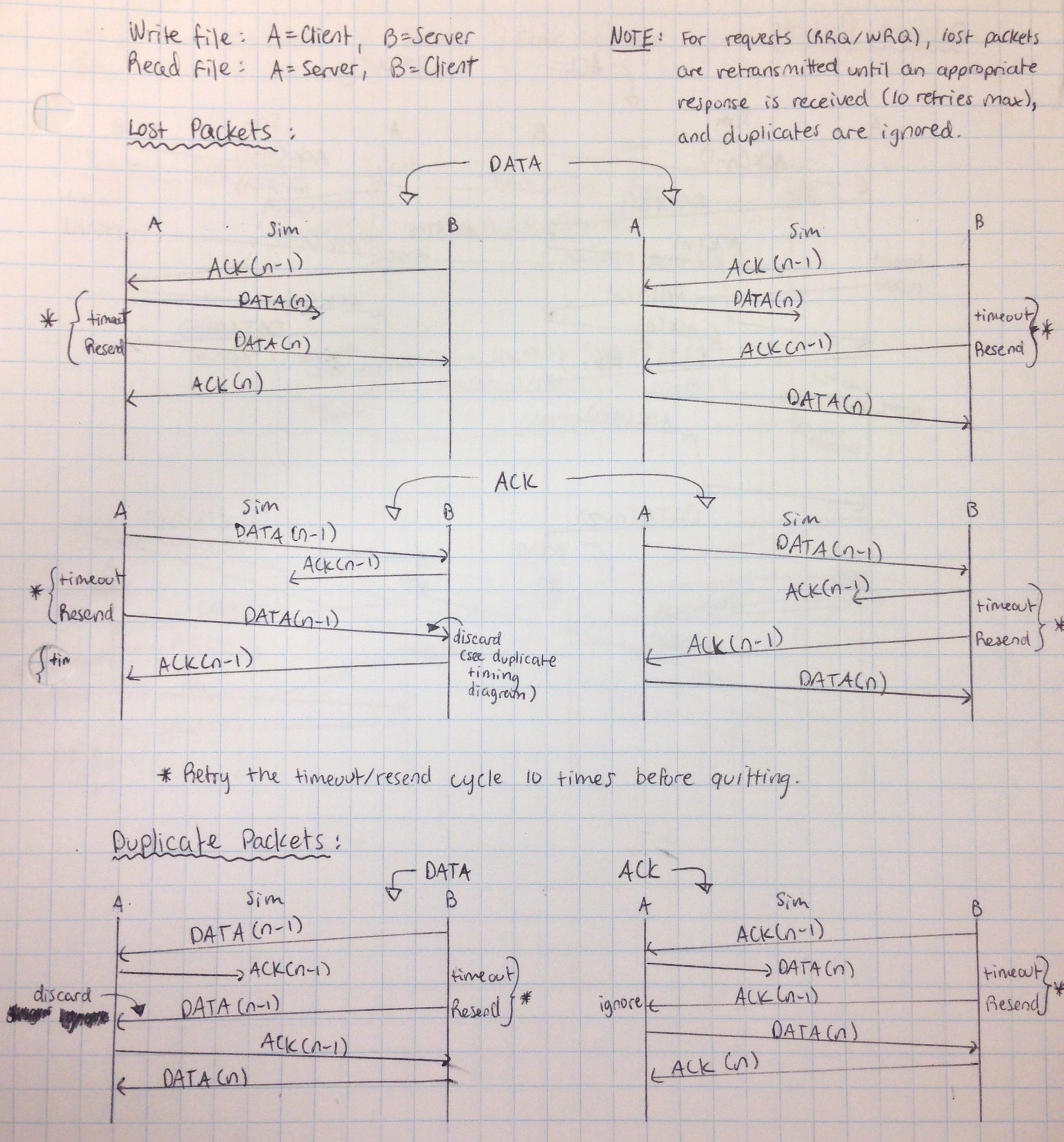
### Normal Operation:



### Delayed Packets:

****

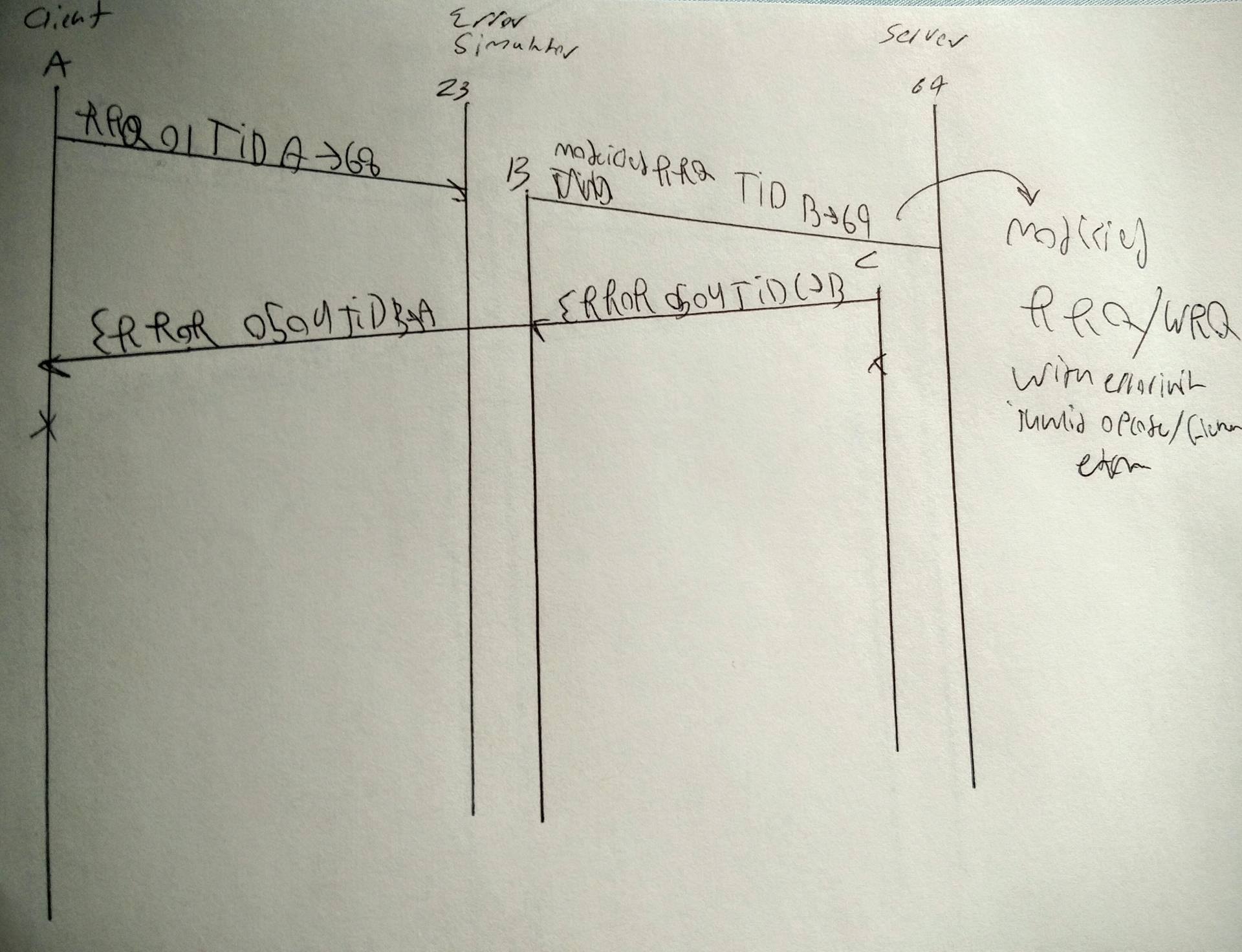
### Lost and Duplicate packets:



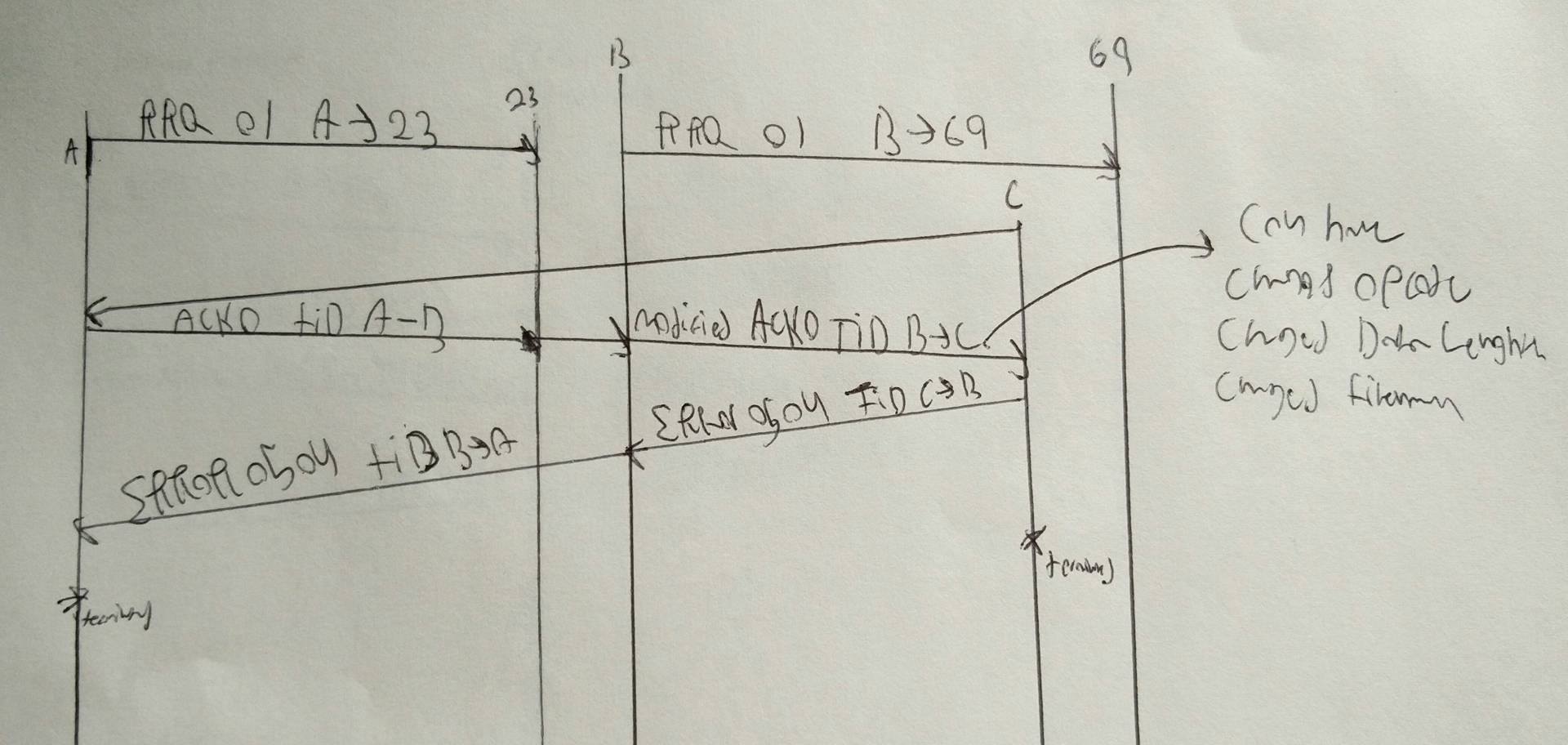
**ERRORS:**

Invalid TID (05): **C:\Users\kinan\AppData\Local\Microsoft\Windows\INetCache\Content.Word\TimingDiagram_Error5_InvalidTID.JPG**

### Invalid Request (04):

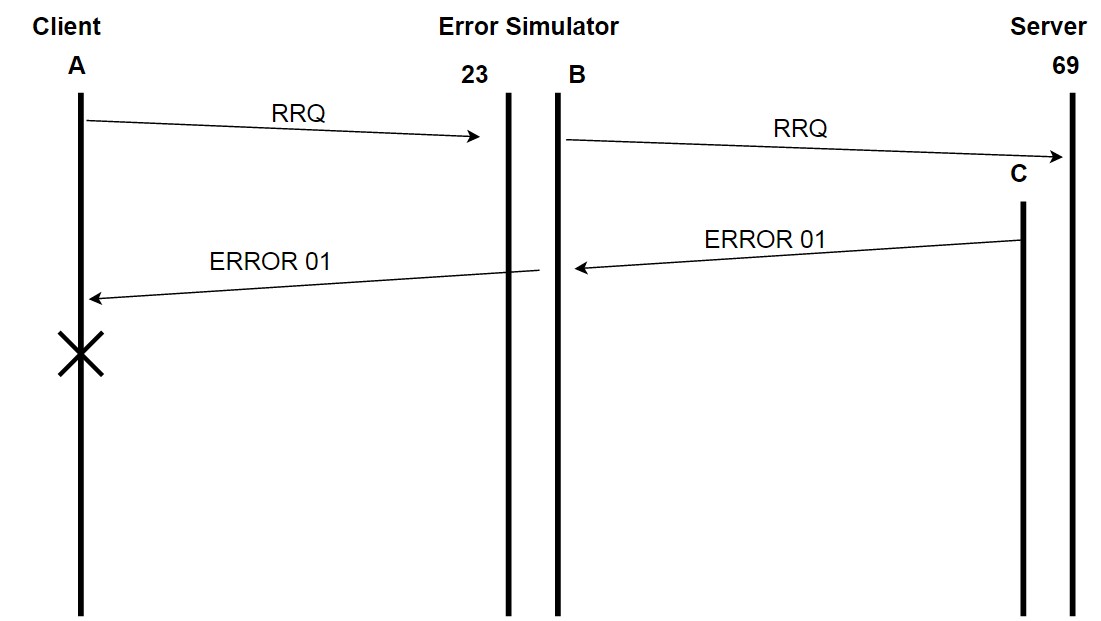


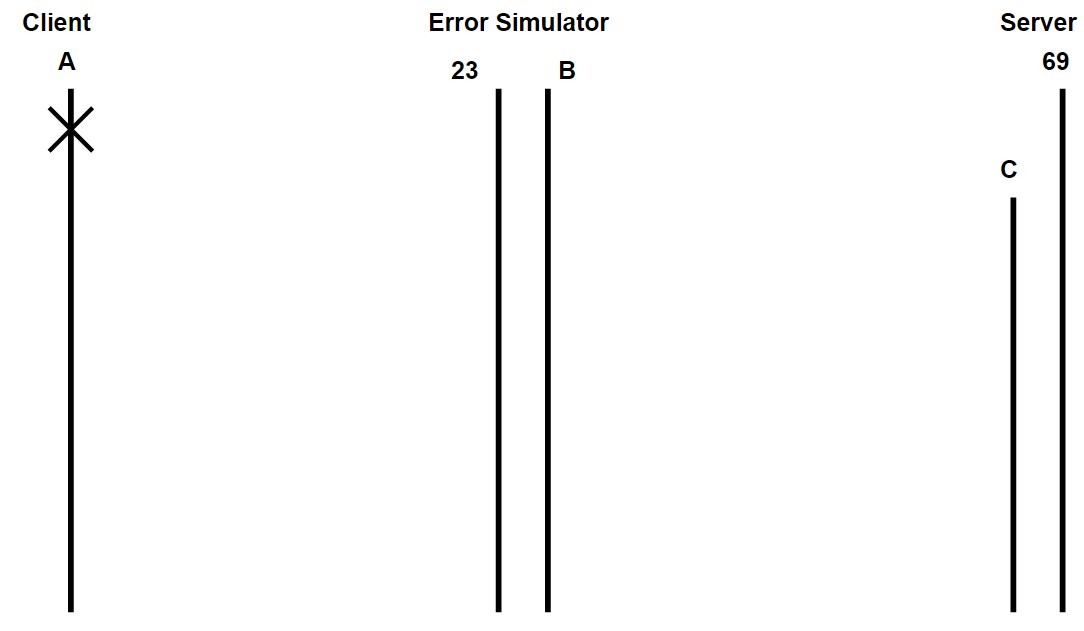
### Invalid Data (04):



### File Not Found (01):

**Read**

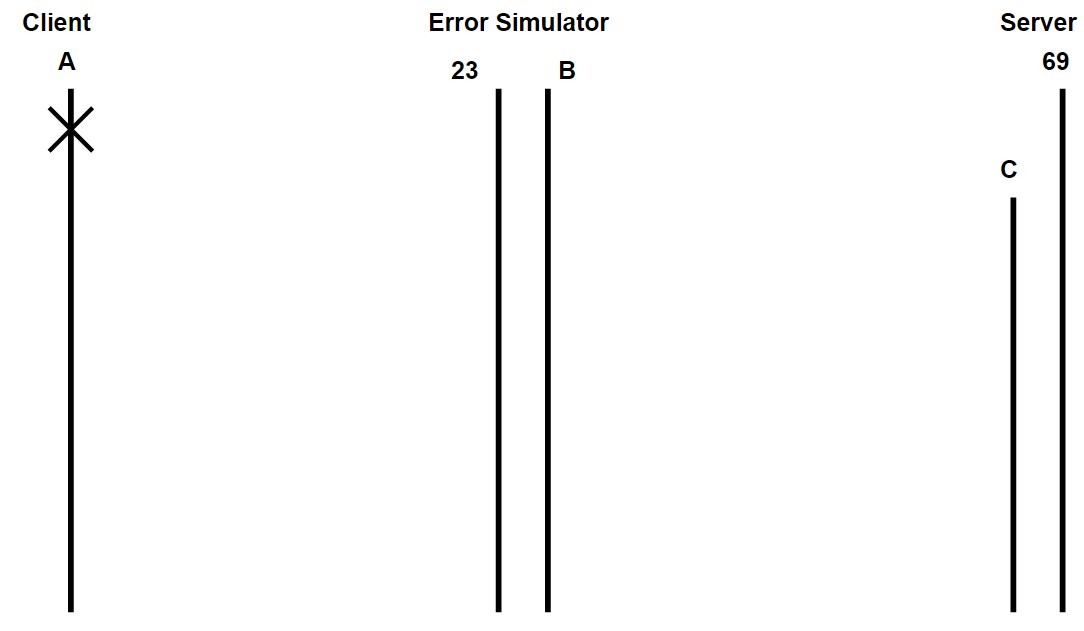


**Write**

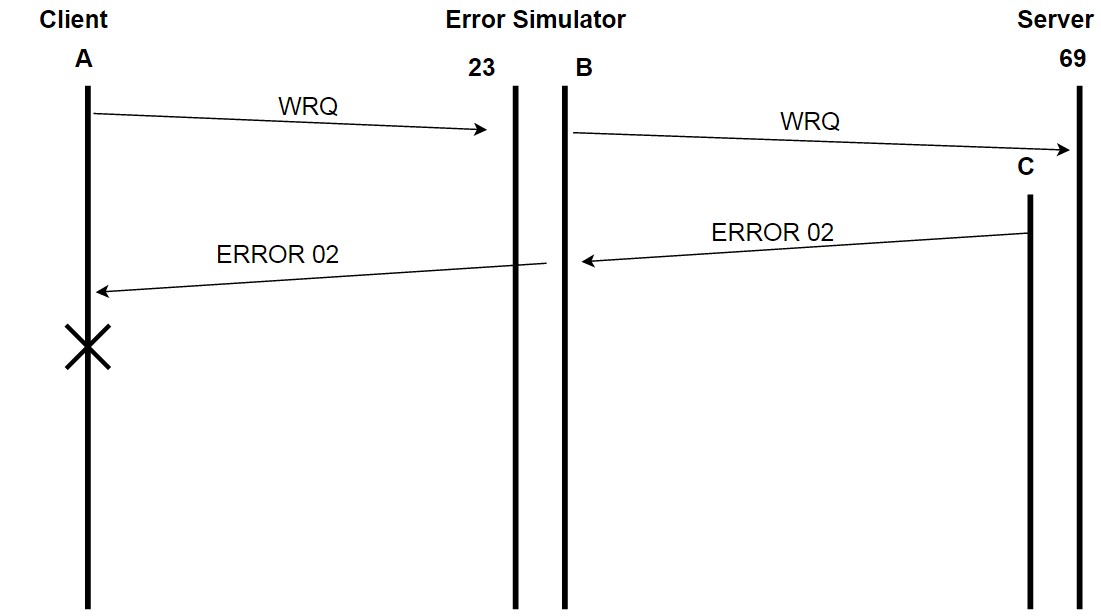
### Access Violation (02)

**READ**

**WRITE from no access file:**

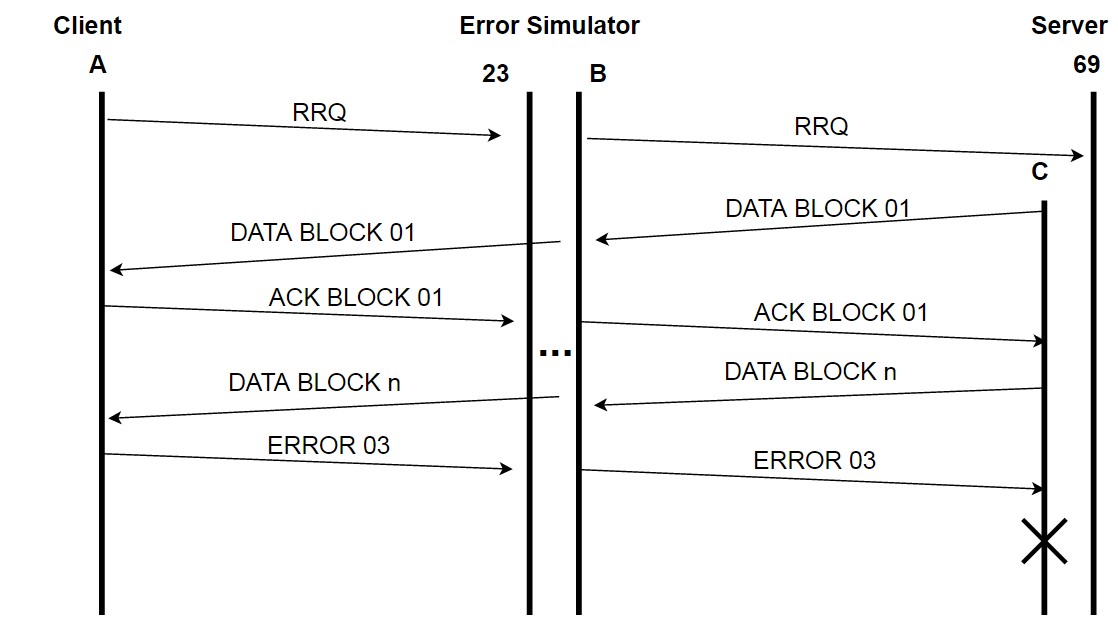


**WRITE to no access file**

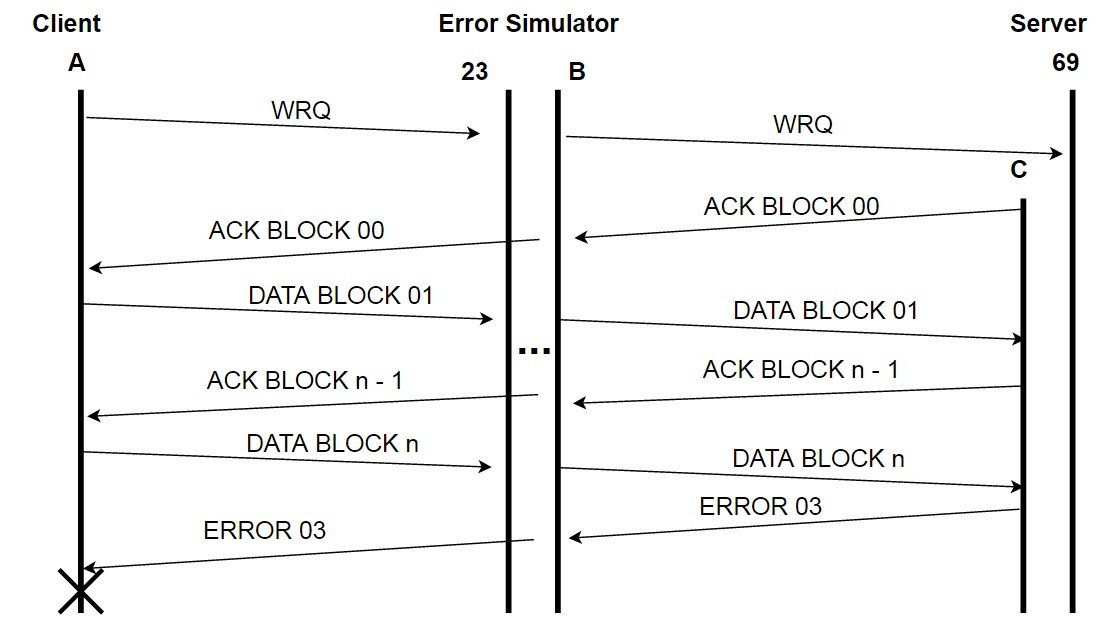


### Disk Full (03)

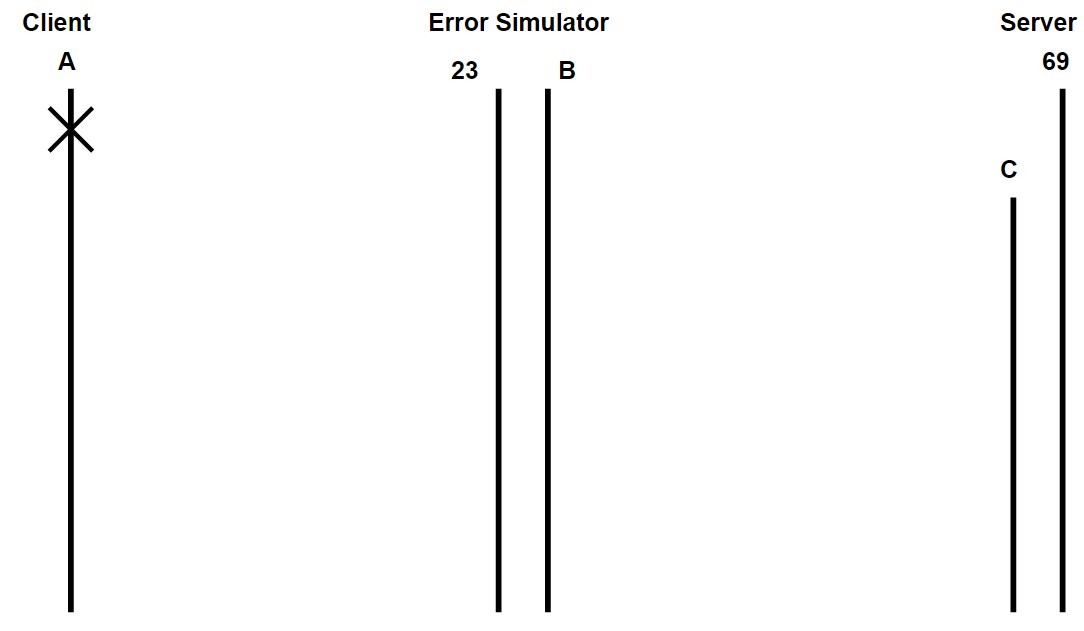
**READ**



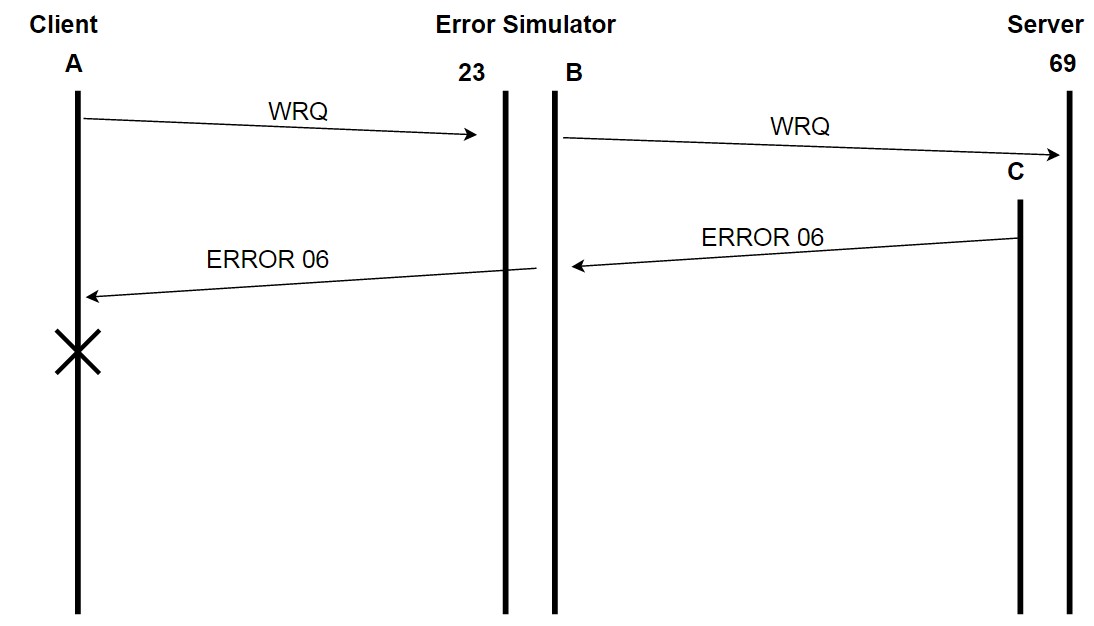
**WRITE**



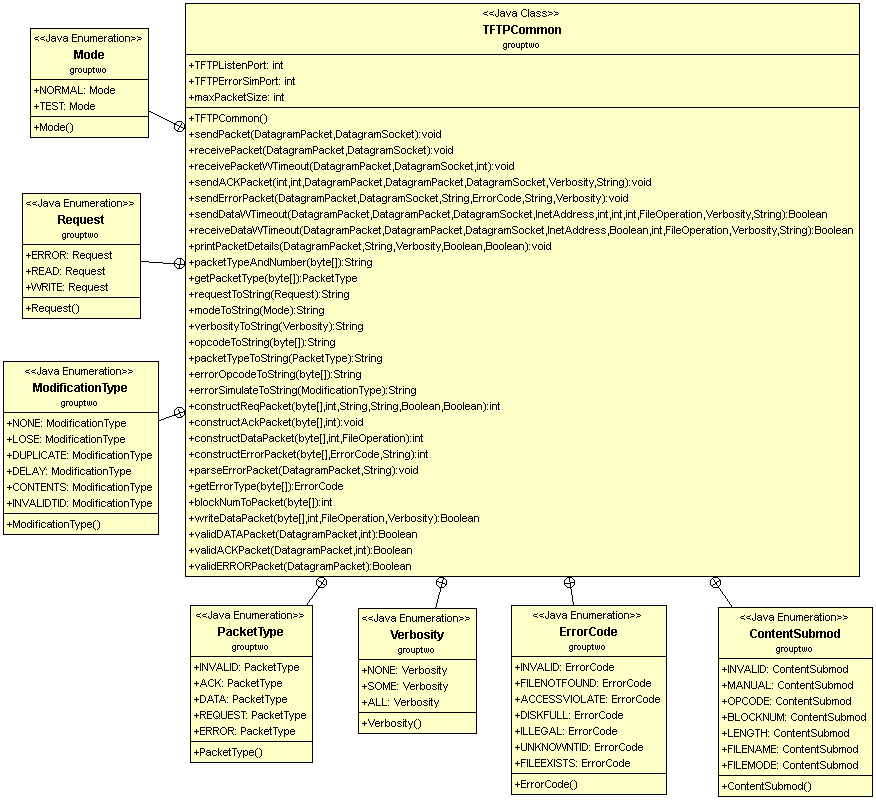
### File Already Exists (06)

**READ**

**WRITE**



# UML Class Diagrams:



# Instructions:

## Setup Instructions

1. Create new Eclipse project
2. Create new Eclipse package in project named “grouptwo”
3. Add all .java files (see below) to a grouptwo folder in your Eclipse workspace

* ClientConnectionThread.java
* FileOperation.java
* TFTPClient.java
* TFTPIntHost.java
* TFTPServer.java
* TFTPCommon.java

1. Run TFTPServer.class
2. Run TFTPIntHost.class
3. Run TFTPClient.class

## Testing Instructions

**TFTPServer**

Enter “o” to set file overwrite

* Enter “true” if you would like to overwrite existing files, and “false” other wise

Enter ”v” to set the verbosity

* Enter “none”, “some”, or “all” to select the verbosity of the output

Enter “t” to set the retransmission timeout

* Enter an integer value (in ms)

Enter ”q” to quit

**TFTPIntHost (Error Simulator)**

Enter “contents” to modify a packets contents

* Enter “request” to modify a WRQ/RRQ packet (chosen by client)
  + If “opcode” is selected, enter the new opcode of the request
  + If “fname” is selected, enter the new name of the file in the request
  + If “mode” is selected, enter the new mode of the file
  + If “rmfname” is selected the file name will be removed from the request
  + If “rmmode” is selected the mode will be removed from the request
  + If “length” is selected, enter the new length of the request (0-516)
  + If “manual” is selected, enter the specific position in the byte array of the request to modify, then enter the new value of that position
  + If “r” is selected, you will be returned to the main menu without saving any changes
  + If “s” is selected, you will be returned to the main menu and your changes will be saved
* Enter “data” to modify a DATA packet or “ack” to modify an ACK packet
  + Enter the number of the packet in the transfer you’d like to modify
  + Can then choose from “opcode”, ”mode”, “length”, “manual”, “r”, and “s” as above
* Enter “error” to modify an ERROR packet
  + Enter the error packet type to modify (first ERROR of this type will be modified)
  + Can then choose from “opcode”, “num” (ERROR type), “length”, “manual”, “r”, and “s” as above
* Enter “r” to return to the main menu

Enter “delay” to delay a packet, or “dup” to duplicate a packet

* Enter “request”, “data”, “ack”, or “error” to choose the packet type or r to return to the main menu
  + If “data”, “ack”, or “error” is selected, first enter the number of the packet in the transfer you’d like to modify
* Enter the time between duplicated packets or the delay time (depending on if you’re duplicating the packet or delaying it)

Enter “lose” to lose a packet

* Enter “request”, “data”, “ack”, or “error” to choose the packet type or r to return to the main menu
  + If “data”, “ack”, or “error” is selected, enter the number of the packet in the transfer you’d like to lose

Enter “tid” to send a packet with an Invalid TID

* Enter  “data”, “ack”, or “error” to choose the packet type or r to return to the main menu (request is blocked, can’t have invalid TID on a request packet)
  + Enter the packet number to send from an unknown TID

Enter “c” to cancel a modification that will be made to a packet

Enter “i” to set the IP address of the server (defaulted to local machine)

* Enter server IP address or enter nothing for local host

Enter “p” to print the modifications that will be made to a packet

Enter ”v” to set the verbosity

* Enter “none”, “some”, or “all” to select the verbosity of the output

Enter ”q” to quit

**TFTPClient**

Enter “i” to set the IP address of the intermediate host (defaulted to local machine)

* Enter intermediate host IP address or nothing for localhost

Enter ”m” to set the mode:

* Enter “test” to send through the Error Simulator
* Enter “normal” to send directly to the server

Enter ”o” to set the overwrite preference

* Enter “true” if you wish to overwrite files on the client, “false” to disable overwriting.

Enter ”v” to set the verbosity

* Enter “none”, “some”, or “all” to select the verbosity of the output

Enter “t” to set the retransmission timeout

* Enter an integer value (in ms)

Enter ”q” to quit

To start a request enter a request in one of the following formats (file names must be surrounded by quote):

* read “**readFile.txt”** to “**dest.txt”**
* write “**writeFile.txt”** to “**dest.txt”**

Where the first file must exist for reads and writes

## Test Cases

|  |  |  |
| --- | --- | --- |
| **Test Files** | **Reason** | **Result** |
| 0.dat | To test an empty file | Successful |
| 1000.dat | File between 512 and 1024 bytes | Successful |
| 1024.dat | Multiple of 512 bytes | Successful |
| 50000.dat | Arbitrary bigger file | Successful |
| 512.dat | 512 bytes (TFTP data packet limit) | Successful |
| Max.dat | To test the max size (65535 \* 512 bytes) | Successful |
| test.txt |  | Successful |
| toomuch.dat | To test more than max size | Successful - bytes rollover and transfer beyond 65535 packets |

## Additional tests performed:

### Iteration #4

**Tests on RRQ**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Packet**  **Type** | **tid** | **Dup** | **Lose** | **Delay** | **Opcode** | **Block #** | **Packet Length** | **Change Contents** |
| **REQUEST** | Not Possible | GOOD Server handles first request and times out after on the dup RRQ | GOOD  Client resend RRQ after 1000ms | GOOD combination of dup and lose | GOOD Server responds with error 4 and correct error message. Both client and server cancel | Not Possible | GOOD  Request is rejected if packet is missing bytes or is too long (padded with 0s) | GOOD  Error sim doesn’t modify if byte doesn’t exist and error occur correctly if request is malformed |
| **DATA** | GOOD  Client forms error packet 5 and sends it back to the invalid TID | GOOD  Client doesn’t write dup data and sends back an ack for the data. Server ignores the dup ack | GOOD  Server times out waiting for ACK for lost DATA, and resends DATA. Client acks and transfer continues | GOOD  Retransmits if delay > timeout, waits if timeout > delay | GOOD  Server receives error.  Transfer is cancelled on both sides | GOOD  Client send error 4 if block number is higher than expected, both sides end transfer. | GOOD Server receives error. Transfer cancels | GOOD  Data contents get modified and written to file. Errors are sent for bad opcode or block number |
| **ACK** | GOOD  Server forms error 5 and sends it back (ACK times out though with 1 s t/o) | GOOD Server receives dup ACK and ignores it | GOOD Server resends DATA after it doesn’t receive the ACK, Client ignores the now duplicated DATA | GOOD  Same effect as delaying DATA | GOOD Client receives error. Transfer is cancelled on both sides | GOOD  Server sends error 4 if block number is high than expected, both sides end transfer. Lower block number acts as a delay | GOOD  Server sends error 4 is the length is too small or too large | GOOD  Errors are sent the same way as bad opcode or blocknum |
| **ERROR** | GOOD  Forced error 4 on a data packet, and invalid TID on the error. Client shuts down from the bad data packet and server sends error 5 and times out | GOOD Client and Server transfer threads end, so dup ERROR is never received | GOOD  Forcing error 4 on data causes client to send error and end transfer. Server doesn’treceive it and retransmits data until it times out and quits | GOOD Error is received after delay and server thread shuts down | GOOD  Error sim forced error 4 on data, client sends error 4, server receives error packet with bad opcode and sends another error packet back.  Both sides terminate | GOOD  Error sim forced error 4, then changed error 4 to error 2. Correct error was received and transfer was terminated | GOOD  Increasing length past original works fine, transfer cancels.  Decreasing length to 2 causes receiving end to send another error back | GOOD  Modifying packets error message causes the resulting string to have a different character. Same effect as bad error code and opcode |

**Other Errors**

File deleted if error? GOOD

File not found (Error Code 01) GOOD

Access Violation (Error Code 02)  GOOD, Works on Windows

Disk Full (Error Code 03) GOOD, on OS X and Windows. Part file is deleted when error is sent

File Already Exists (Error Code 06) GOOD, client doesn’t start transferring if localFile exists

**Tests on WRQ**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Packet**  **Type** | **tid** | **Dup** | **Lose** | **Delay** | **Opcode** | **Block #** | **Packet Length** | **Change Contents** |
| **REQUEST** | Not possible | 2 Threads created, client detects two acks, ignores one, other thread dies | GOOD client times out resends wrq | GOOD | GOOD se  nds back error 4 and terminates transfer | N/A | GOOD  Less:  se  nds back error 4 and terminates transfer  More:  sends back error 4 and terminates transfer | GOOD tested changing a byte in the file name and the server returns file not found |
| **DATA** | GOOD server sends error 5 back to same port | GOOD duplicate data not written to file and ack sent again | Good client sends data again | Good client sends data again | Server sends error 4 | GOOD  For both smaller and bigger block numbers | Less than 4: server error 4  More than 4: server thinks it received last packet and finishes transfer, client retransmits  Bigger i.e 999  good error 04 | Good writes changes to to file |
| **ACK** | GOOD client sends error 05 to port that sent ack, times out after 1s and retransmits data | GOOD  Client receives and ignores duplicate ack | Good client sends data again | Good client sends data again | GOOD clients sends error 04 and quits, server receives and quits | Bigger Client sends error 4  to server  Smaller: GOOD | GOOD  Less than 4: sends error 4  More than 4:  Client sends error 4 | GOOD |
| **ERROR** | GOOD  Side receiving invalid TID error responds with error 5 and times out. Side that sent original error already stopped | Good client cancels before it receives duplicate error  NOTE: packet number is the error code | OK error sent twice from server because it received request twice, client exits upon receival of error | GOOD | GOOD  Error is modified, receiving end send another error 4 because it received an invalid packet  Both sides end | GOOD  Error type is changed by error sim. Same behaviour to bad opcode | GOOD  Length = 1  OK server dies client keeps sending data then dies  Length 2 =  error still client quits GOOD  Length 999  Longer i.e 999 GOOD | GOOD messed up the message in the error packet by adding a random byte |

**Other Errors**

File deleted if error? GOOD

File not found (Error Code 01) - client will print error message and will not send request

Access Violation (Error Code 02)- GOOD

Disk Full (Error Code 03) - starts and stops to send error 03 when disk is full

File Already Exists (Error Code 06) - GOOD sends error 6

### Iteration #5

**Tests on RRQ**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Packet**  **Type** | **tid** | **Dup** | **Lose** | **Delay** | **Opcode** | **Block #** | **Packet Length** | **Change Contents** |
| **REQUEST** | Not Possible | GOOD  Duplicate request packet is ignored before ClientConnectionThread creation | GOOD Same as iteration 4 | OK not GOODIf the request gets resent because of delay, original CCC dies (?) transfer is OK though | GOOD Same as iteration 4 | Not Possible | GOOD Same as iteration 4 | GOOD same as iteration 4 |
| **DATA** | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as  iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 |
| **ACK** | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 |
| **ERROR** | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | Not possible | GOOD same as iteration 4 | GOOD same as iteration 4 |

**Other Errors**

File deleted if error? GOOD same as iteration 4

File not found (Error Code 01) GOOD same as iteration 4

Access Violation (Error Code 02) GOOD same as iteration 4

Disk Full (Error Code 03) GOOD same as iteration 4

File Already Exists (Error Code 06) GOOD same as iteration 4

**Tests on WRQ**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Packet**  **Type** | **tid** | **Dup** | **Lose** | **Delay** | **Opcode** | **Block #** | **Packet Length** | **Change Contents** |
| **REQUEST** | Not possible | GOOD same as iteration 4 | GOOD same as iteration 4 | OK, same problem as read | GOOD same as iteration 4 | Not Possible | GOOD same as iteration 4 | GOOD same as iteration 4 |
| **DATA** | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 |
| **ACK** | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 |
| **ERROR** | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 | GOOD same as iteration 4 |

**Other Errors**

File deleted if error? GOOD same as iteration 4

File not found (Error Code 01) GOOD same as iteration 4

Access Violation (Error Code 02) GOOD same as iteration 4

Disk Full (Error Code 03)  GOOD same as iteration 4

File Already Exists (Error Code 06) GOOD same as iteration 4