TFTP Implementation

Project SYSC 3303 - Real-Time Concurrent Systems

Griffin Barrett,#100978435 ;
Brydon Gibson, #100975274;
Lisa Martini, #101057495 .

December 5, 2016

Contents

1 Responsabilities per Iteration

1.1 Iteration 1 and 2

For the 2 first iterations, all three of us were in different groups Griffin in group 15, Brydon in group 5 and Lisa in group 17. We did most of our previous group's work, but Lisa and Brydon had troubles with their former group and IT2 hasn't been correctly submitted.

We decided to start our collaboration with Griffin's codebase because it was the only codebase that implemented IT2.

1.2 Iteration 3

The most difficult part of this iteration was to understand and be more familiar with this new code and refactor it to follow the specification correctly. Then the goal of this iteration was to implement Network Error handling.

Brydon Gibson (100975274) :

- wrote the interpreter for the control language for the intermediate host and the associated explanations in ReadMe.
- tested TFTP transfers and errors with intermediate host.
- started to refactor Sender to deal with timeout/retransmission.

Griffin Barrett(100978435) :

- wrote code for Receiver to deal with timeout/retransmission
- finished Sender code
- wrote Client and Server code to deal with timeout/retransmission.

Lisa Martini (101057495) :

- drew the timing diagrams and UML CLass diagrams
- wrote ReadMe
- wrote code to deal with error loss or delay in sender and receiver
- took care of verbose/quiet mode printing correctly

1.3 Iteration 4

In this iteration we needed to handle TFTP error .

Brydon Gibson (100975274) :

- Wrote the intermediate host code to provoke error 4 and error 5 packet and the associated explanations in ReadMe.
- Testing
- Added the new conditions to the intermediate host language interpreter,
- Drew Timing diagrams for error 5.

Griffin Barrett(100978435) :

- Drew Timing diagrams for Error 4 and UML class diagrams.
- Helped Brydon with Intermediate Host. timeout/retransmission.
- Implemented strict packet parsing with informative error messages.

Lisa Martini (101057495) :

- Wrote code for handling (creating/receiving) TFTP errors in Server, Client, Sender and Receiver.
- Refactored packets package to check if packets followed specification.
- Wrote ReadMe

1.4 Iteration 5

For this iteration we needed to implement the TFTP File transfer on different computers and set everything for final presentation.

Brydon Gibson (100975274) :

- Fixed Intermediate host.
- Full testing of the system.
- Small bugfixes and refactoring

Griffin Barrett(100978435) :

- Implemented removing files after a failed transfer.
- Modified UCM diagram
- Updates class diagrams
- Created test files and test file genorator
- Added test file generation to the server and client
- Final edit of this document.

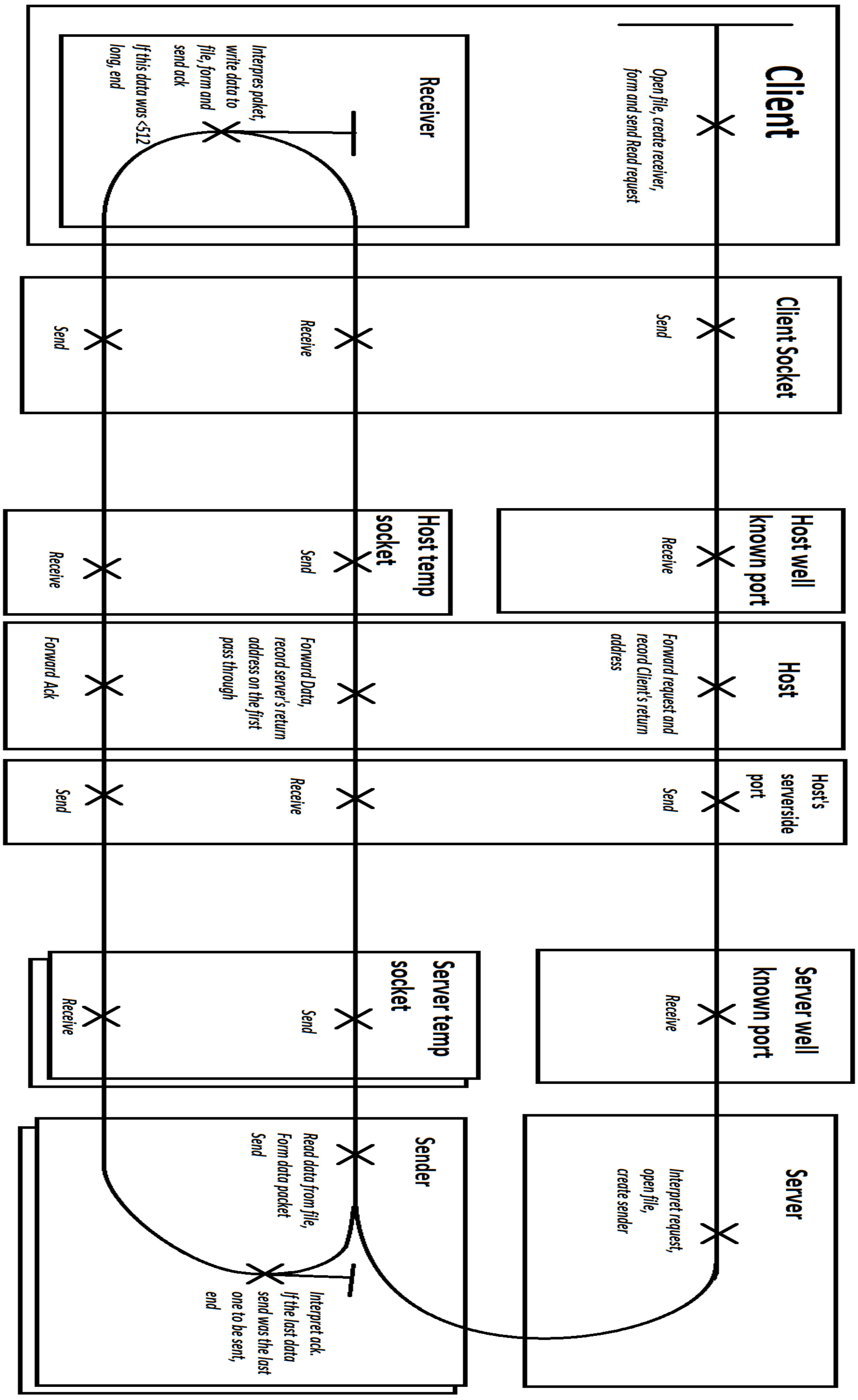
Lisa Martini (101057495) :

- Wrote code to allow client to set/change server location to transfer a file from a different computer.
- Wrote final report.
- Redrew diagrams from IT2 and IT3 to follow the current state of the implementation

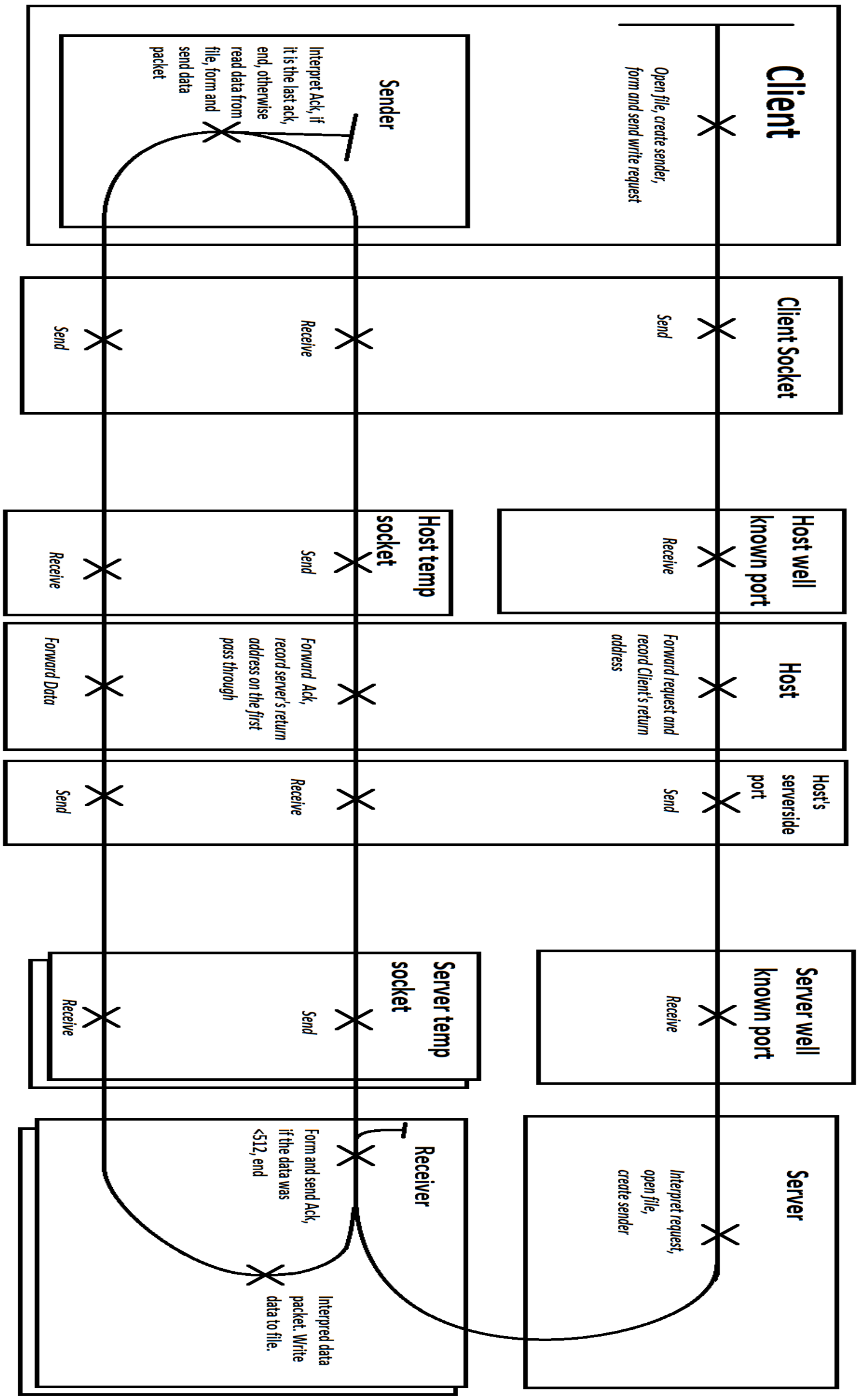
2 Diagrams

2.1 UCMs for read and write

2.1.1 UCM read request


Client
Open file, create receiver, form and send Read request
Receiver
Interpres paket, write data to file, form and send ack
If this data was <512 long, end
Client Socket
Send
Receive
Send
Host well known port
Receive
Host temp socket
Send
Receive
Host
Forward request and record Client's return address
Forward Data, record server's return address on the first pass through
Forward Ack
Host's serverside port
Send
Receive
Send
Server well known port
Receive
Server temp socket
Send
Receive
Server
Interpret request, open file, create sender
Sender
Read data from file, Form data packet Send
Interpret ack. If the last data send was the last one to be sent, end

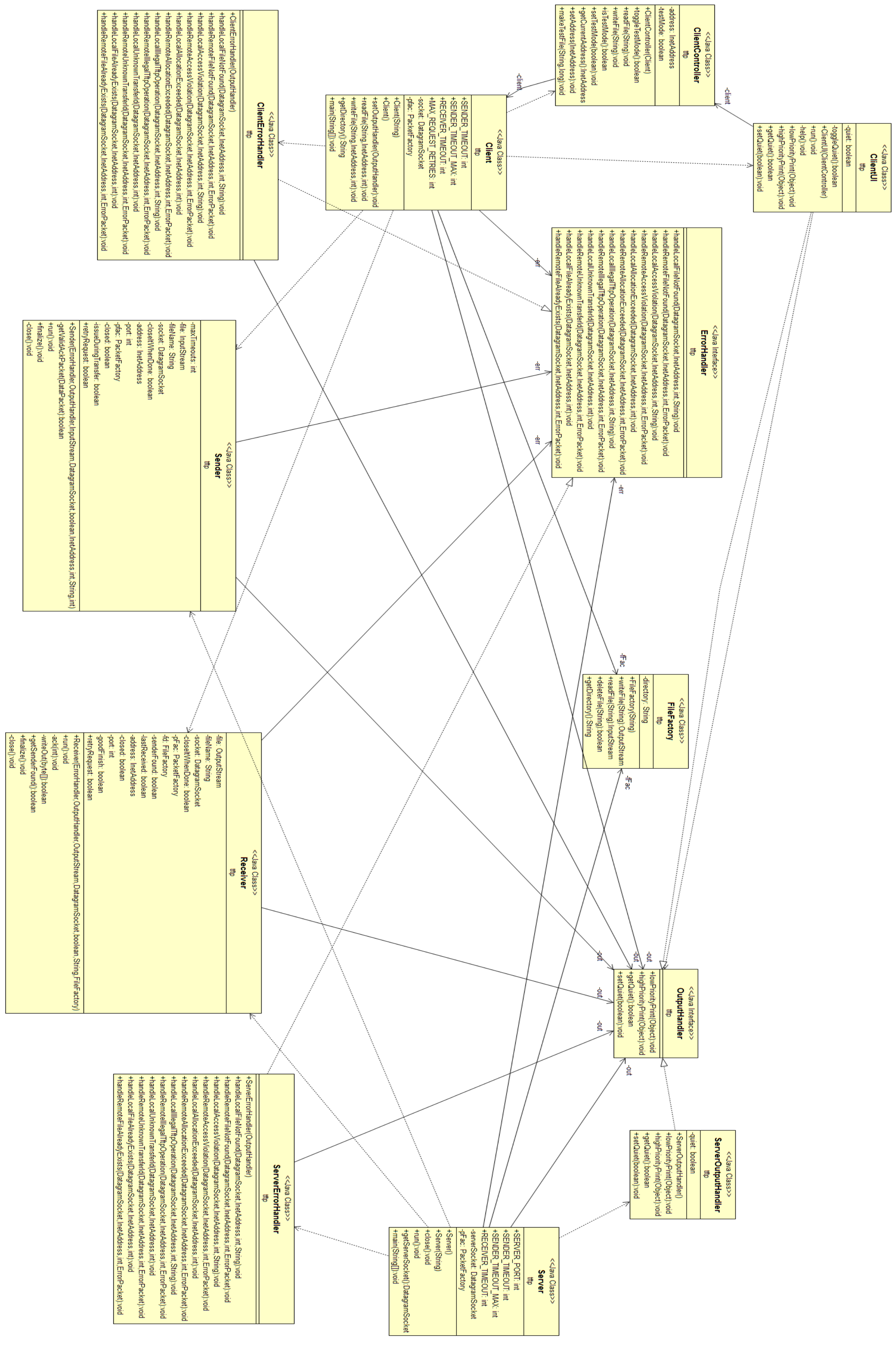
2.1.2 UCM write request



Client
Open file, create sender, form and send write request
Sender
Interpret Ack, if it is the last ack, end, otherwise read data from file, form and send data packet
Client Socket
Send
Receive
Send
Host well known port
Receive
Host temp socket
Send
Receive
Host
Forward request and record Client's return address
Forward Ack, record server's return address on the first pass through
Forward Data
Host's serverside port
Send
Receive
Send
Server well known port
Receive
Server temp socket
Send
Receive
Server
Interpret request, open file, create sender
Receiver
Form and send Ack, if the data was <512, end
Interpred data packet. Write data to file.

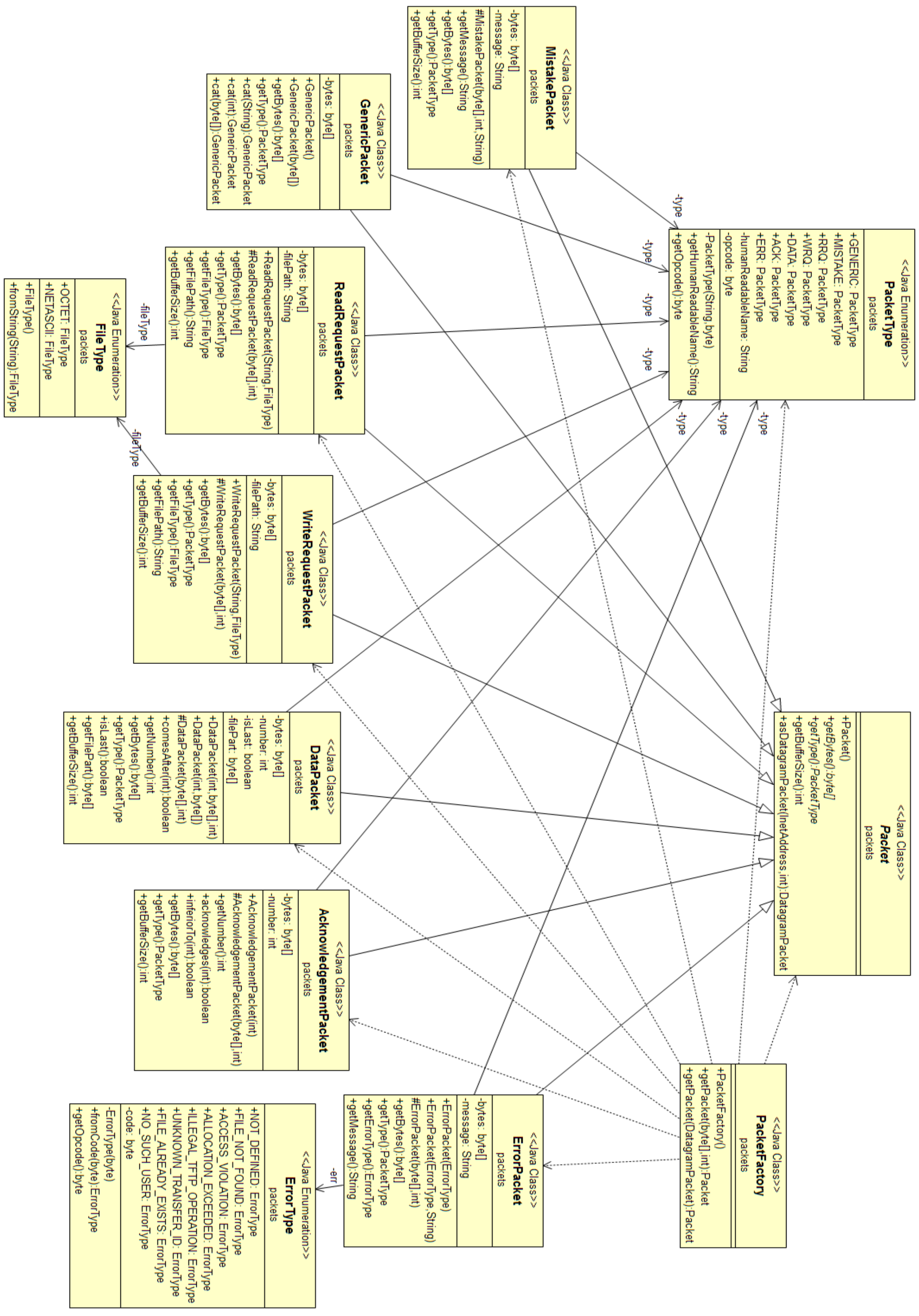
2.2 UML Class Diagrams

2.2.1 TFTP Package

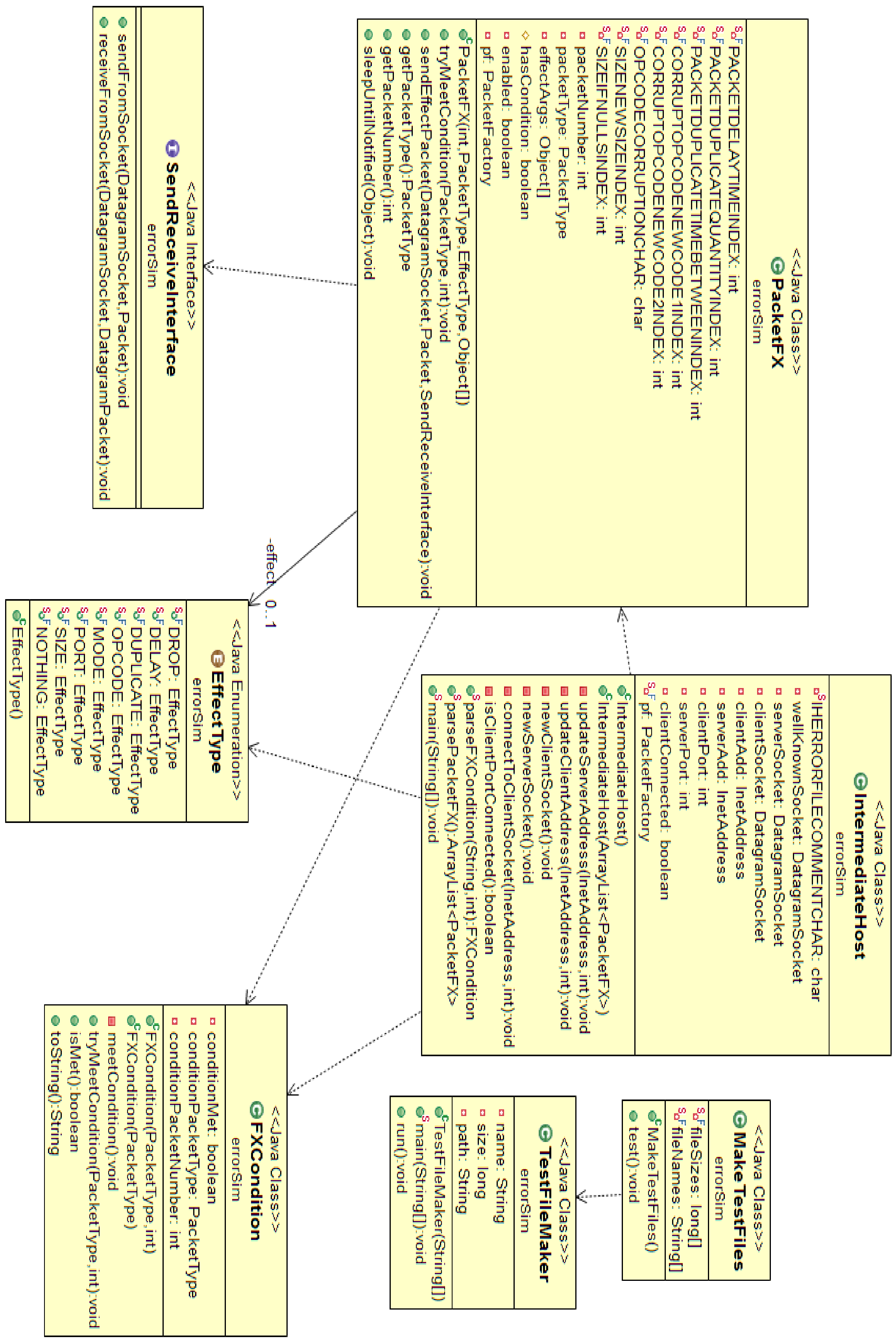


<<Java Class>>
ClientUI
tftp
-quiet: boolean
-toggleQuiet():boolean
+ClientUI(ClientController)
+run():void
-help():void
+lowPriorityPrint(Object):void
+highPriorityPrint(Object):void
+getQuiet():boolean
+setQuiet(boolean):void
-client
<<Java Class>>
ClientController
tftp
-address: InetAddress
-testMode: boolean
+ClientController(Client)
+toggleTestMode():boolean
+readFile(String):void
+writeFile(String):void
+isTestMode():boolean
+setTestMode(boolean):void
+getCurrentAddress():InetAddress
+setAddress(InetAddress):void
+makeTestFile(String,long):void
-client
<<Java Interface>>
ErrorHandler
tftp
+handleLocalFileNotFound(DatagramSocket,InetAddress,int,String):void
+handleRemoteFileNotFound(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalAccessViolation(DatagramSocket,InetAddress,int,String):void
+handleRemoteAccessViolation(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalAllocationExceeded(DatagramSocket,InetAddress,int):void
+handleRemoteAllocationExceeded(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalIllegalTftpOperation(DatagramSocket,InetAddress,int,String):void
+handleRemoteIllegalTftpOperation(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalUnknownTransferId(DatagramSocket,InetAddress,int):void
+handleRemoteUnknownTransferId(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalFileAlreadyExists(DatagramSocket,InetAddress,int):void
+handleRemoteFileAlreadyExists(DatagramSocket,InetAddress,int,ErrorPacket):void
-err
-err
-err
-err
<<Java Class>>
FileFactory
tftp
-directory: String
+FileFactory(String)
+writeFile(String):OutputStream
+readFile(String):InputStream
+deleteFile(String):boolean
+getDirectory():String
-fFac
-fFac
<<Java Interface>>
OutputHandler
tftp
+lowPriorityPrint(Object):void
+highPriorityPrint(Object):void
+getQuiet():boolean
+setQuiet(boolean):void
-out
-out
-out
-out
-out
-out
<<Java Class>>
ServerOutputHandler
tftp
-quiet: boolean
+ServerOutputHandler()
+lowPriorityPrint(Object):void
+highPriorityPrint(Object):void
+getQuiet():boolean
+setQuiet(boolean):void
<<Java Class>>
Client
tftp
+SENDER_TIMEOUT: int
+SENDER_TIMEOUT_MAX: int
+RECEIVER_TIMEOUT: int
+MAX_REQUEST_RETRIES: int
-socket: DatagramSocket
-pfac: PacketFactory
+Client(String)
+Client()
+setOutputHandler(OutputHandler):void
+readFile(String,InetAddress,int):void
+writeFile(String,InetAddress,int):void
+getDirectory():String
+main(String[]):void
<<Java Class>>
Server
tftp
+SERVER_PORT: int
+SENDER_TIMEOUT: int
+SENDER_TIMEOUT_MAX: int
+RECEIVER_TIMEOUT: int
-serverSocket: DatagramSocket
-pFac: PacketFactory
+Server()
+Server(String)
+close():void
+run():void
+getServerSocket():DatagramSocket
+main(String[]):void
<<Java Class>>
ClientErrorHandler
tftp
+ClientErrorHandler(OutputHandler)
+handleLocalFileNotFound(DatagramSocket,InetAddress,int,String):void
+handleRemoteFileNotFound(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalAccessViolation(DatagramSocket,InetAddress,int,String):void
+handleRemoteAccessViolation(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalAllocationExceeded(DatagramSocket,InetAddress,int):void
+handleRemoteAllocationExceeded(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalIllegalTftpOperation(DatagramSocket,InetAddress,int,String):void
+handleRemoteIllegalTftpOperation(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalUnknownTransferId(DatagramSocket,InetAddress,int):void
+handleRemoteUnknownTransferId(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalFileAlreadyExists(DatagramSocket,InetAddress,int):void
+handleRemoteFileAlreadyExists(DatagramSocket,InetAddress,int,ErrorPacket):void
<<Java Class>>
Sender
tftp
-maxTimeouts: int
-file: InputStream
-fileName: String
-socket: DatagramSocket
-closeItWhenDone: boolean
-address: InetAddress
-port: int
-pfac: PacketFactory
-closed: boolean
-issueDuringTransfer: boolean
+retryRequest: boolean
+Sender(ErrorHandler,OutputHandler,InputStream,DatagramSocket,boolean,InetAddress,int,String,int)
-getValidAckPacket(DataPacket):boolean
+run():void
+finalize():void
-close():void
<<Java Class>>
Receiver
tftp
-file: OutputStream
-fileName: String
-socket: DatagramSocket
-closeItWhenDone: boolean
-pFac: PacketFactory
-fd: FileFactory
-senderFound: boolean
-lastReceived: boolean
-address: InetAddress
-closed: boolean
-port: int
-goodFinish: boolean
+retryRequest: boolean
+Receiver(ErrorHandler,OutputHandler,OutputStream,DatagramSocket,boolean,String,FileFactory)
+run():void
-ack(int):void
-writeOut(byte[]):boolean
+getSenderFound():boolean
+finalize():void
-close():void
<<Java Class>>
ServerErrorHandler
tftp
+ServerErrorHandler(OutputHandler)
+handleLocalFileNotFound(DatagramSocket,InetAddress,int,String):void
+handleRemoteFileNotFound(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalAccessViolation(DatagramSocket,InetAddress,int,String):void
+handleRemoteAccessViolation(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalAllocationExceeded(DatagramSocket,InetAddress,int):void
+handleRemoteAllocationExceeded(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalIllegalTftpOperation(DatagramSocket,InetAddress,int,String):void
+handleRemoteIllegalTftpOperation(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalUnknownTransferId(DatagramSocket,InetAddress,int):void
+handleRemoteUnknownTransferId(DatagramSocket,InetAddress,int,ErrorPacket):void
+handleLocalFileAlreadyExists(DatagramSocket,InetAddress,int):void
+handleRemoteFileAlreadyExists(DatagramSocket,InetAddress,int,ErrorPacket):void

2.2.2 Packets Package



2.2.3 ErrorSim Package



2.3 Timing Diagrams

2.3.1 I/O Errors

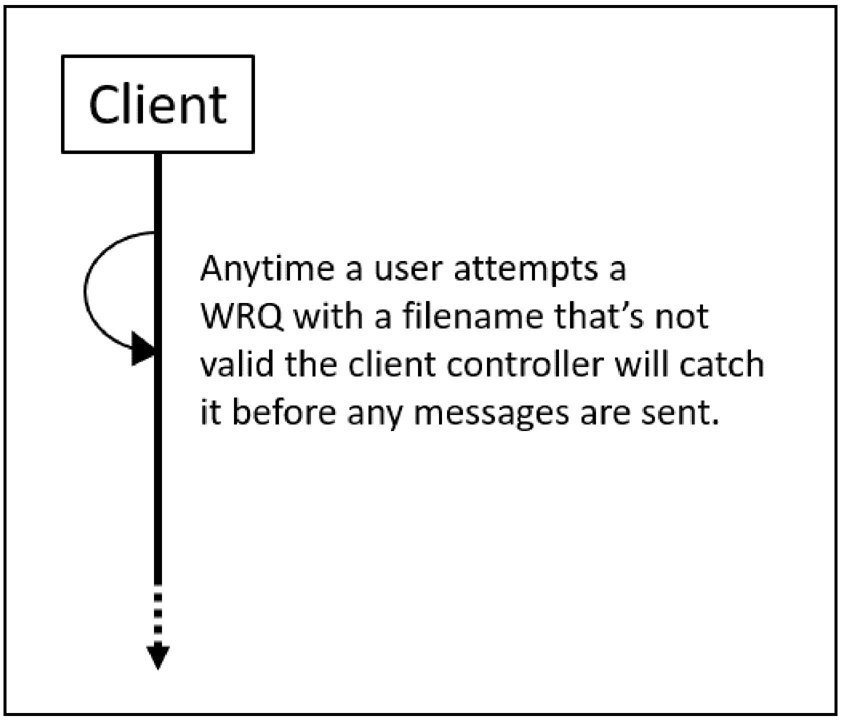

Client
Anytime a user attempts a WRQ with a filename that's not valid the client controller will catch it before any messages are sent.

Figure 1: Client error

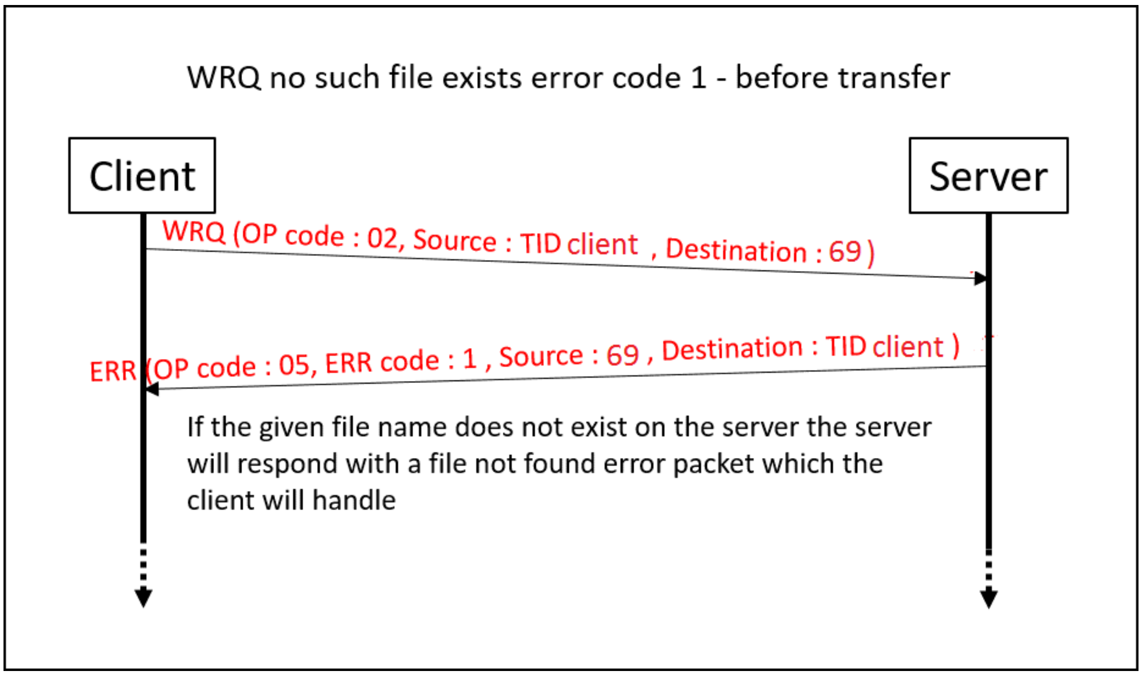

WRQ no such file exists error code 1 - before transfer
Client
Server
WRQ (OP code : 02, Source : TID client , Destination : 69)
ERR (OP code : 05, ERR code : 1 , Source : 69 , Destination : TID client)
If the given file name does not exist on the server the server will respond with a file not found error packet which the client will handle

Figure 2: WRQ error code 1 : file not found

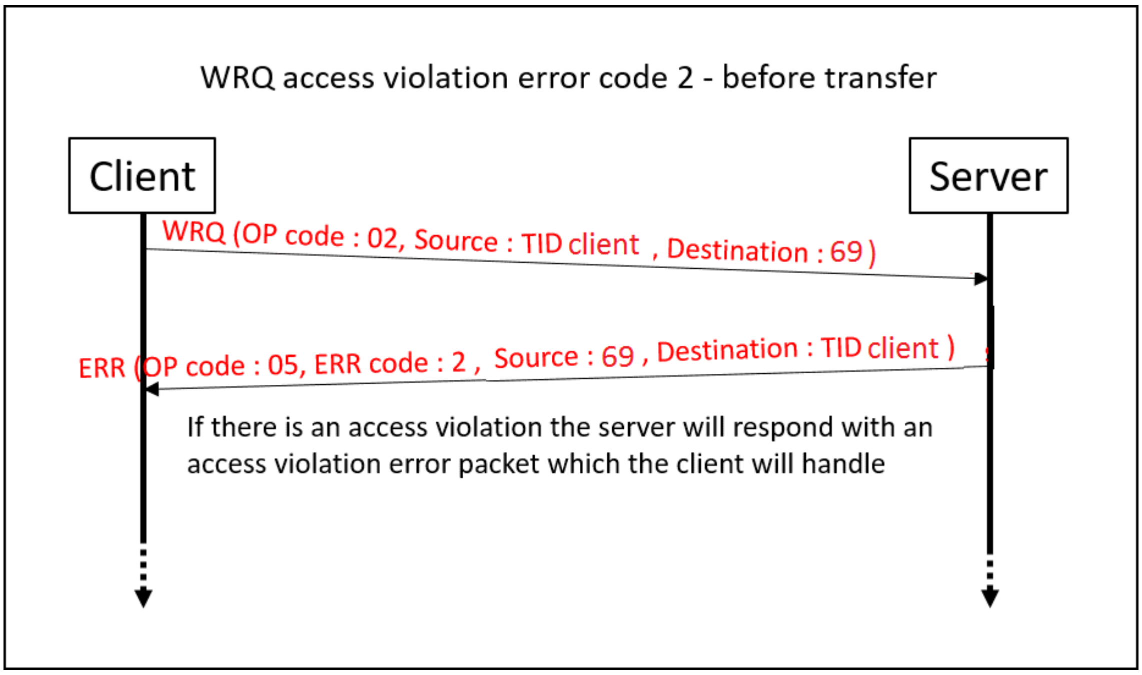

WRQ access violation error code 2 - before transfer
Client
Server
WRQ (OP code : 02, Source : TID client , Destination : 69)
ERR (OP code : 05, ERR code : 2 , Source : 69 , Destination : TID client)
If there is an access violation the server will respond with an access violation error packet which the client will handle

Figure 3: WRQ error code 2 : Access violation

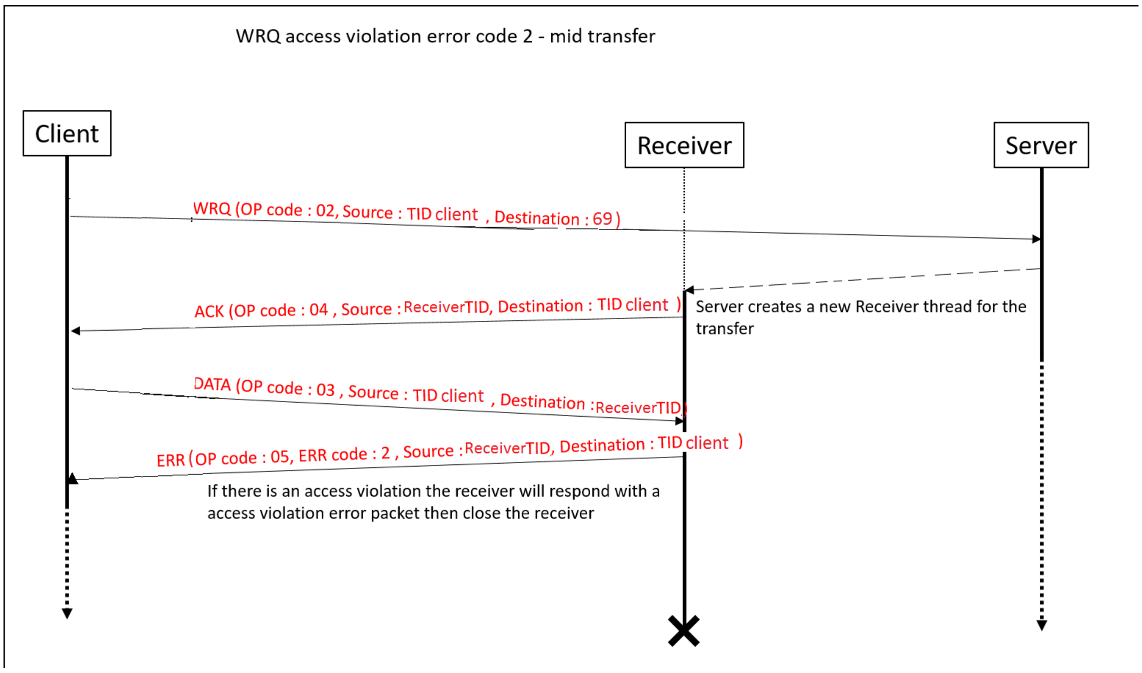

WRQ access violation error code 2 - mid transfer
Client
Receiver
Server
WRQ (OP code : 02, Source : TID client , Destination : 69)
ACK (OP code : 04 , Source : ReceiverTID, Destination : TID client)
Server creates a new Receiver thread for the transfer
DATA (OP code : 03 , Source : TID client , Destination : ReceiverTID)
ERR (OP code : 05, ERR code : 2 , Source : ReceiverTID, Destination : TID client)
If there is an access violation the receiver will respond with a access violation error packet then close the receiver

Figure 4: WRQ error code 2 : Access violation mid-transfer

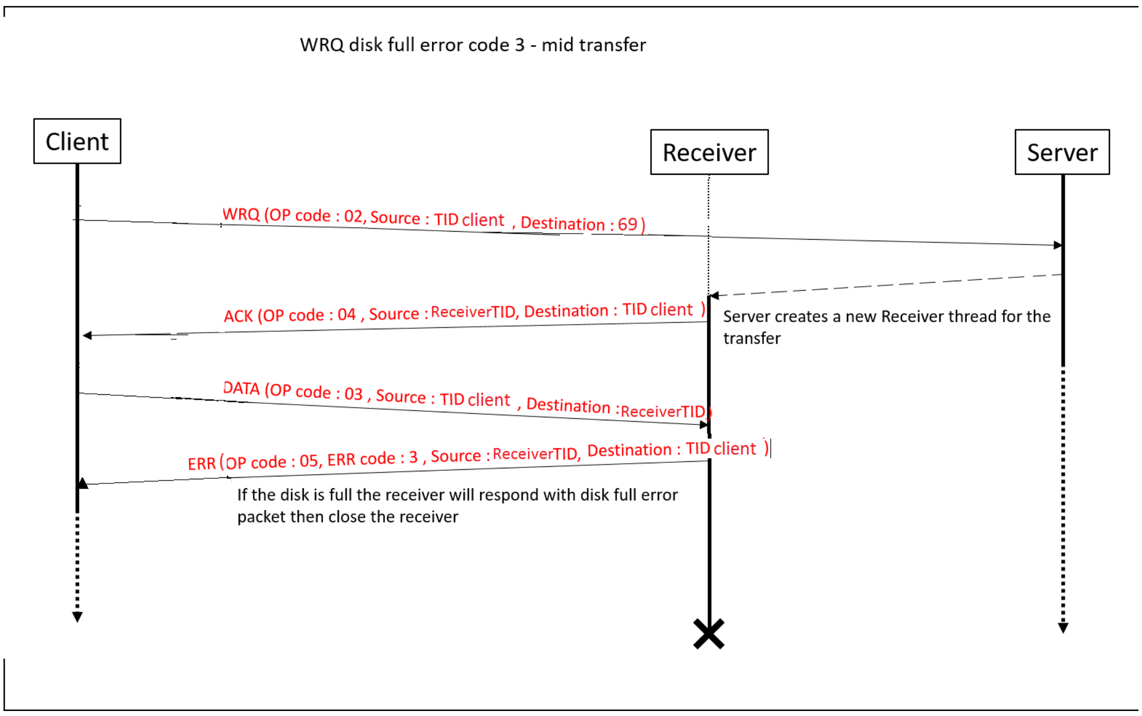

WRQ disk full error code 3 - mid transfer
Client
Receiver
Server
WRQ (OP code : 02, Source : TID client , Destination : 69)
ACK (OP code : 04 , Source : ReceiverTID, Destination : TID client)
Server creates a new Receiver thread for the transfer
DATA (OP code : 03 , Source : TID client , Destination : ReceiverTID)
ERR (OP code : 05, ERR code : 3 , Source : ReceiverTID, Destination : TID client)
If the disk is full the receiver will respond with disk full error packet then close the receiver

Figure 5: WRQ Error code 3, Allocation exceeded

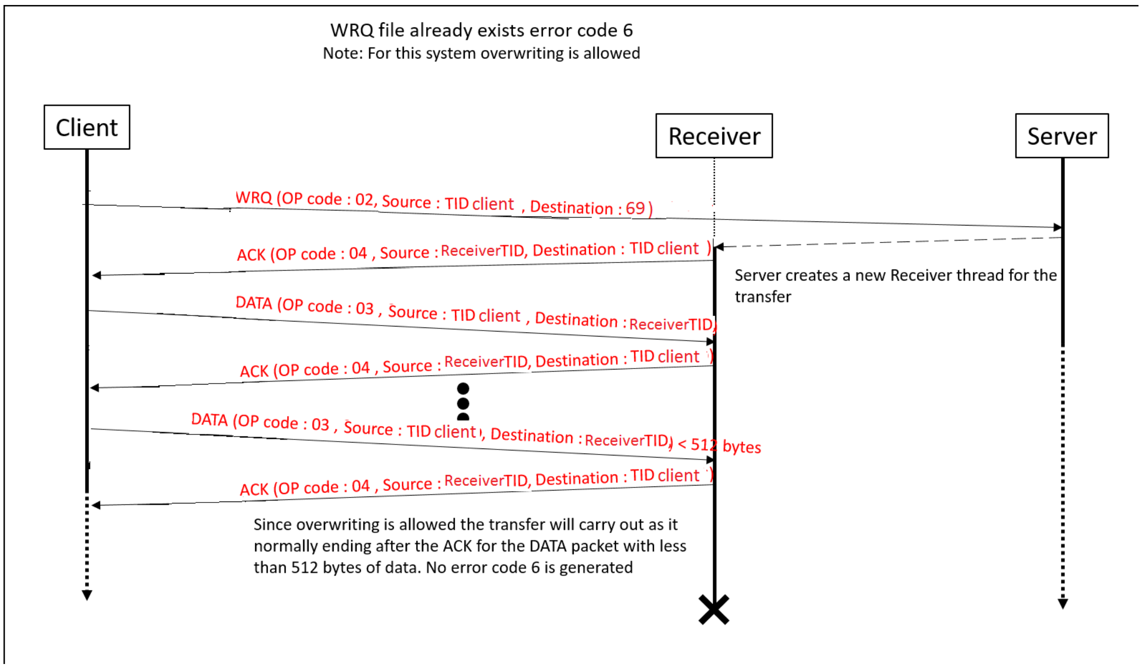

WRQ file already exists error code 6
Note: For this system overwriting is allowed
Client
Receiver
Server
WRQ (OP code : 02, Source : TID client , Destination : 69)
ACK (OP code : 04 , Source : ReceiverTID, Destination : TID client)
Server creates a new Receiver thread for the transfer
DATA (OP code : 03 , Source : TID client , Destination : ReceiverTID)
ACK (OP code : 04 , Source : ReceiverTID, Destination : TID client)
DATA (OP code : 03 , Source : TID client , Destination : ReceiverTID) < 512 bytes
ACK (OP code : 04 , Source : ReceiverTID, Destination : TID client)
Since overwriting is allowed the transfer will carry out as it normally ending after the ACK for the DATA packet with less than 512 bytes of data. No error code 6 is generated

Figure 6: WRQ error code 6 : file already exists

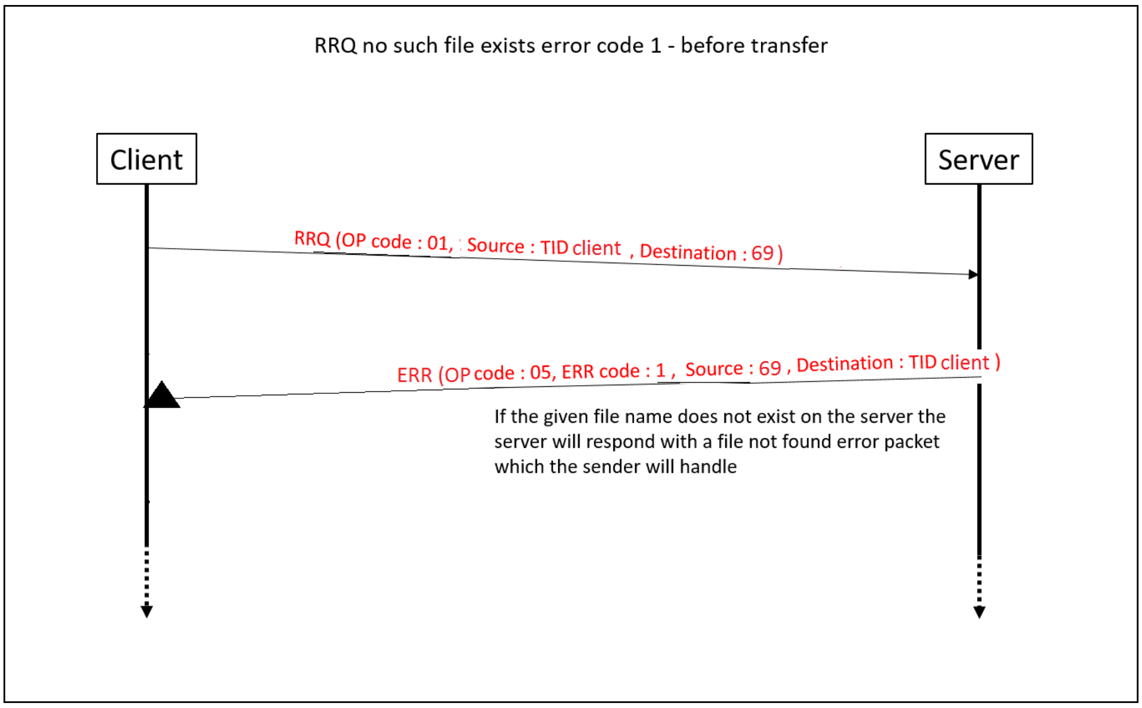

RRQ no such file exists error code 1 - before transfer
Client
Server
RRQ (OP code : 01, Source : TID client , Destination : 69)
ERR (OP code : 05, ERR code : 1 , Source : 69 , Destination : TID client)
If the given file name does not exist on the server the server will respond with a file not found error packet which the sender will handle

Figure 7: RRQ error code 1 : file not found

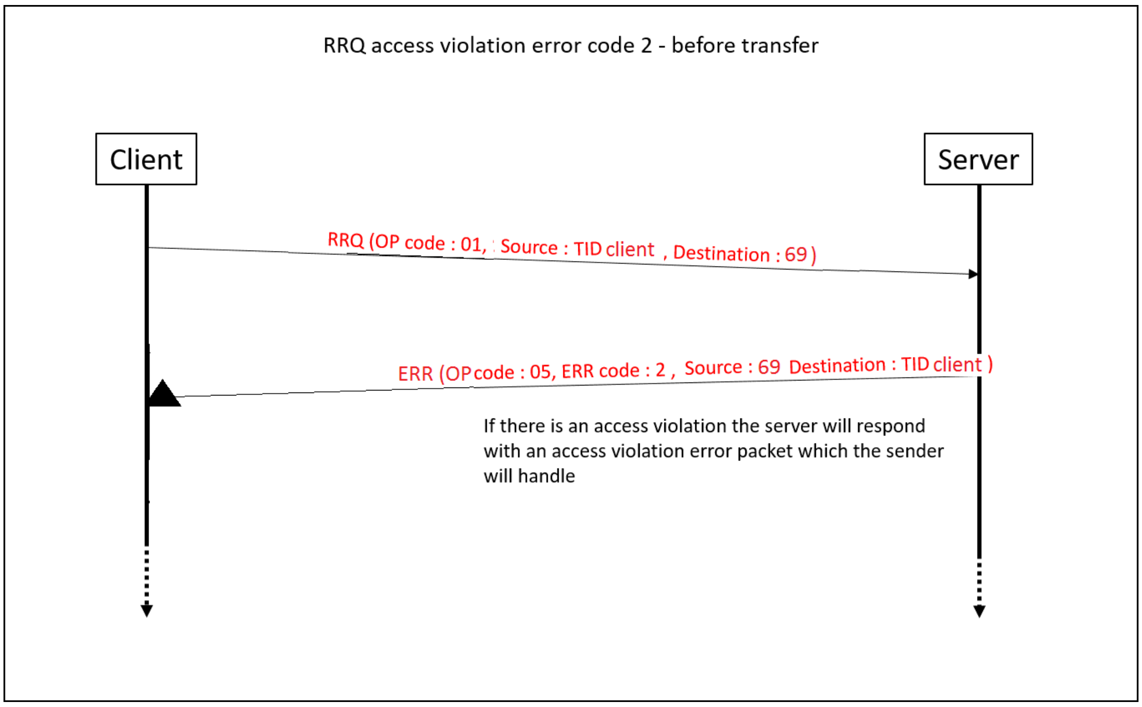

RRQ access violation error code 2 - before transfer
Client
Server
RRQ (OP code : 01, Source : TID client , Destination : 69)
ERR (OP code : 05, ERR code : 2 , Source : 69 Destination : TID client)
If there is an access violation the server will respond with an access violation error packet which the sender will handle

Figure 8: RRQ error code 2 : Access violation

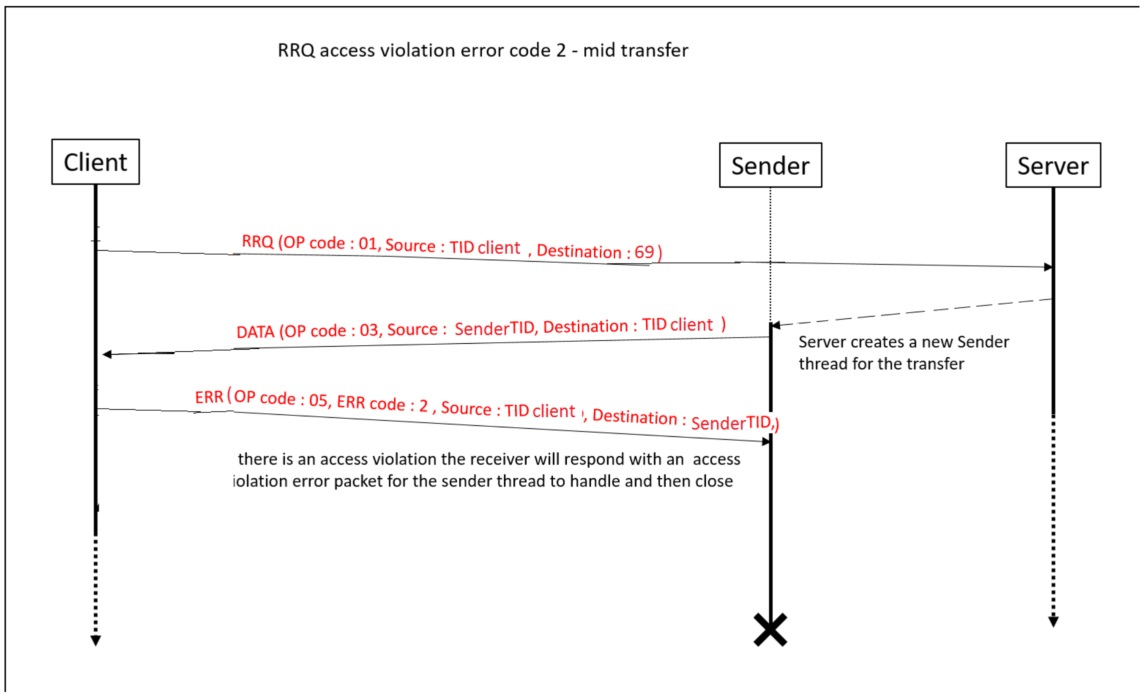

RRQ access violation error code 2 - mid transfer
Client
Sender
Server
RRQ (OP code : 01, Source : TID client , Destination : 69)
DATA (OP code : 03, Source : SenderTID, Destination : TID client)
Server creates a new Sender thread for the transfer
ERR (OP code : 05, ERR code : 2 , Source : TID client , Destination : SenderTID,)
there is an access violation the receiver will respond with an access iolation error packet for the sender thread to handle and then close

Figure 9: RRQ error code 2 : Access violation mid-transfer

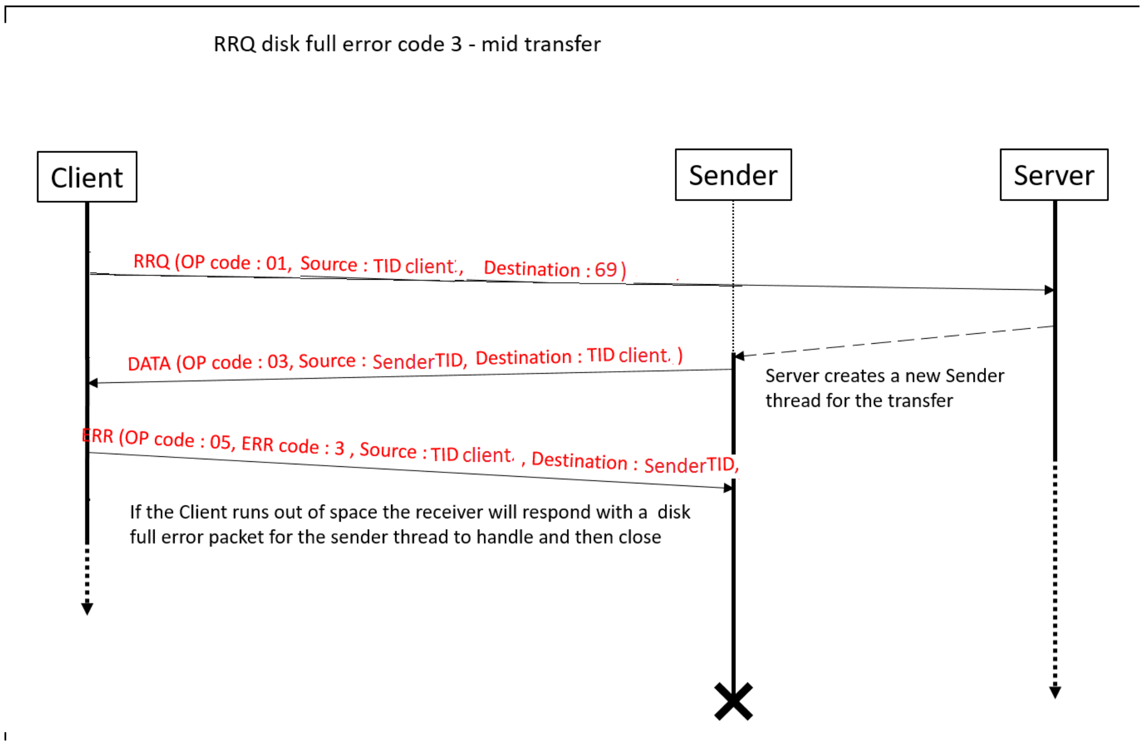

RRQ disk full error code 3 - mid transfer
Client
Sender
Server
RRQ (OP code : 01, Source : TID client, Destination : 69)
DATA (OP code : 03, Source : SenderTID, Destination : TID client.)
Server creates a new Sender thread for the transfer
ERR (OP code : 05, ERR code : 3 , Source : TID client. , Destination : SenderTID,
If the Client runs out of space the receiver will respond with a disk full error packet for the sender thread to handle and then close

Figure 10: RRQ Error code 3, Allocation exceeded

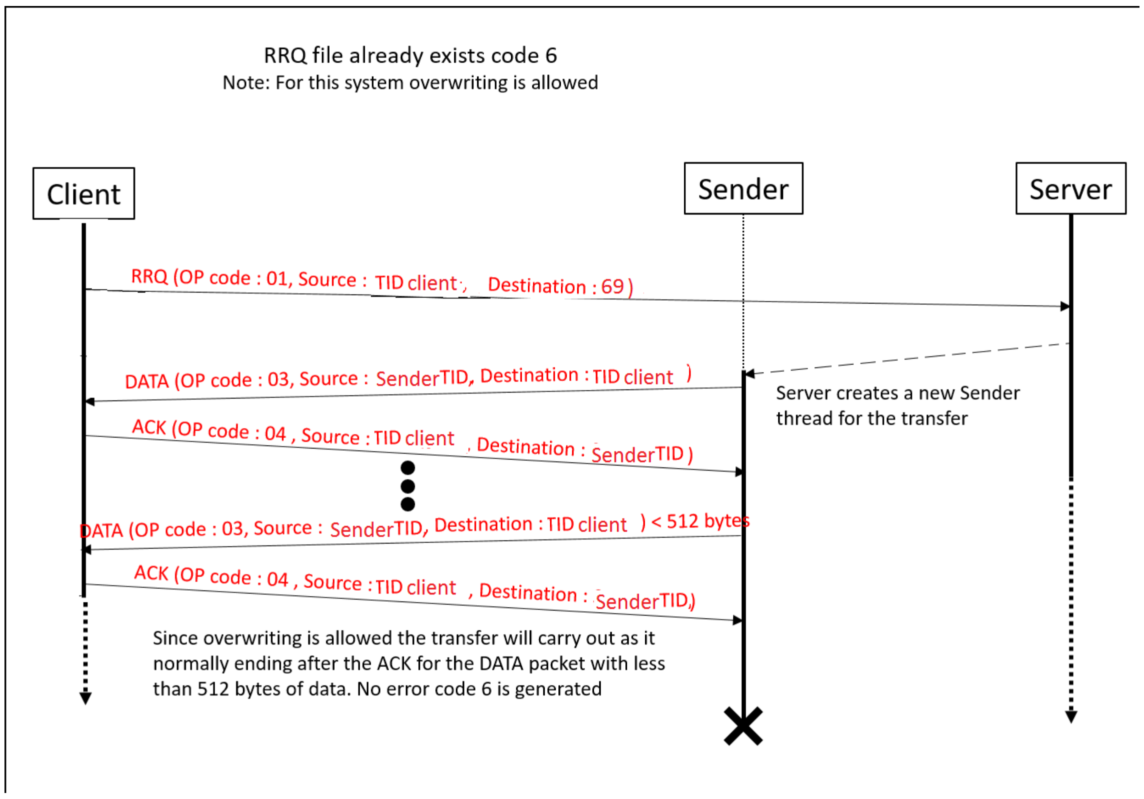

RRQ file already exists code 6
Note: For this system overwriting is allowed
Client
Sender
Server
RRQ (OP code : 01, Source : TID client , Destination : 69)
DATA (OP code : 03, Source : SenderTID, Destination : TID client)
Server creates a new Sender thread for the transfer
ACK (OP code : 04 , Source : TID client , Destination : SenderTID)
DATA (OP code : 03, Source : SenderTID, Destination : TID client) < 512 bytes
ACK (OP code : 04 , Source : TID client , Destination : SenderTID)
Since overwriting is allowed the transfer will carry out as it normally ending after the ACK for the DATA packet with less than 512 bytes of data. No error code 6 is generated

Figure 11: RRQ error code 6 : file already exists

2.3.2 Network Errors

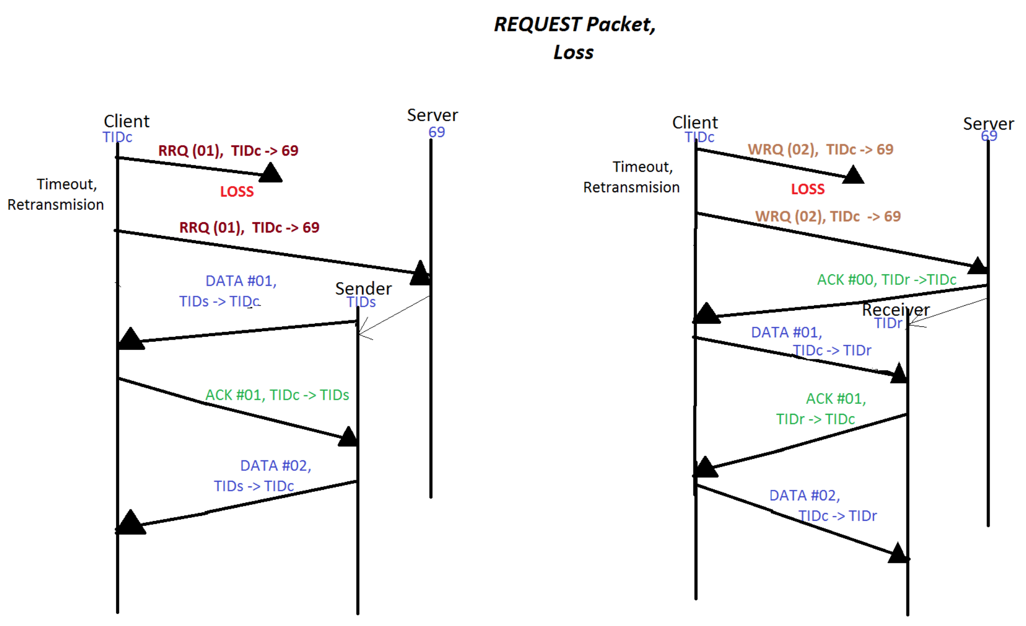


Figure 12: Request packet, loss

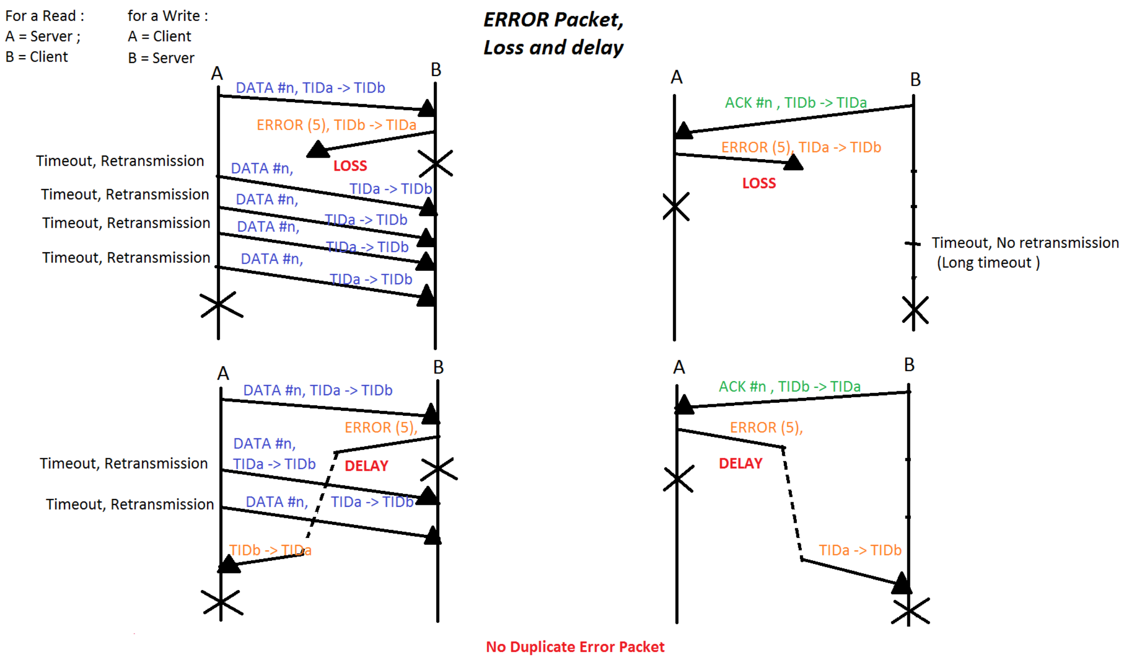


Figure 13: ERROR packet, loss and delay

This previous diagram is valid for all errors except error code 5 (the host shuts only if receives this error but not the one who sent it)

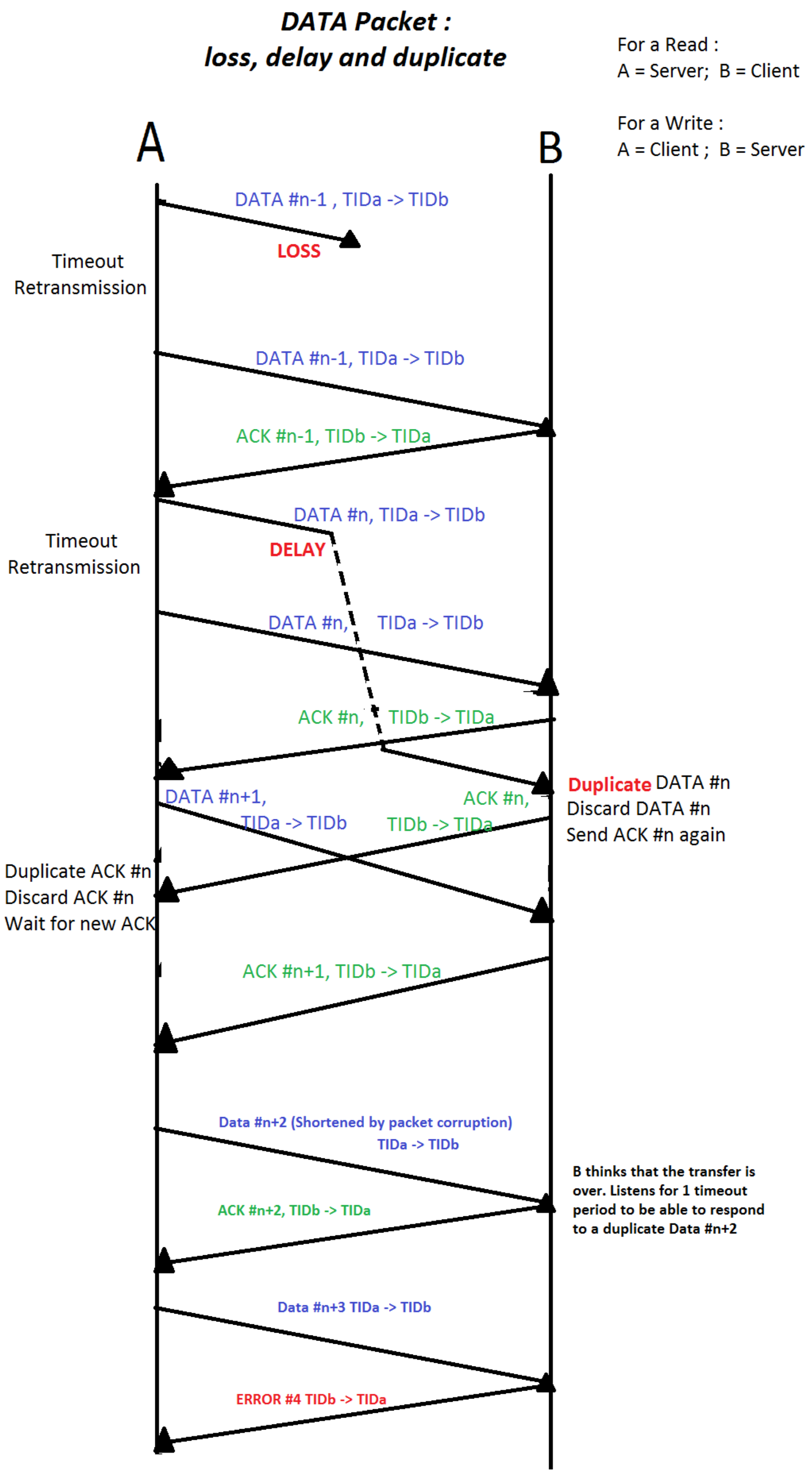


Figure 14: DATA packet, loss, delay, duplicate and shorten

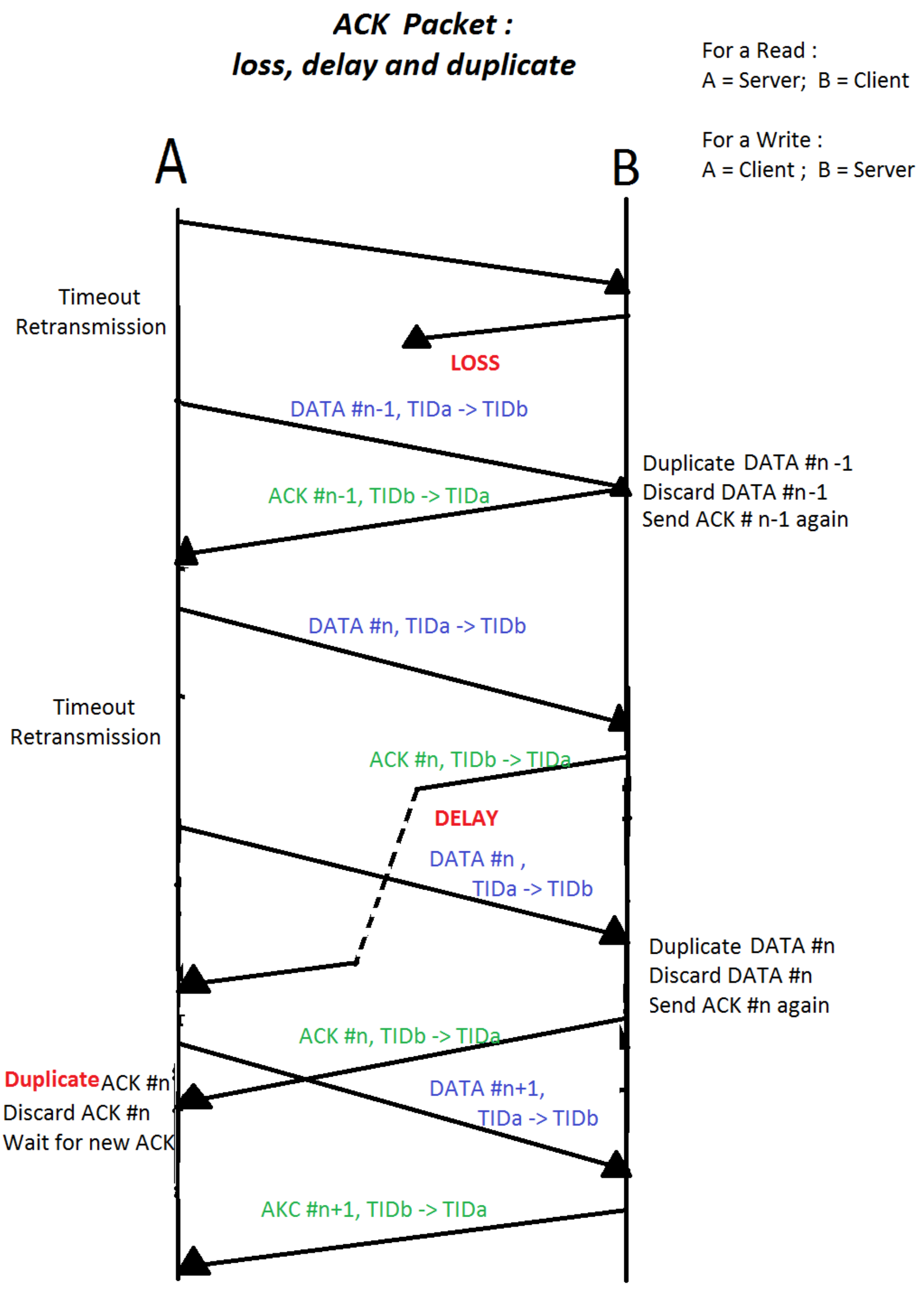


Figure 15: ACK packet, loss, delay and duplicate

2.3.3 TFTP Errors

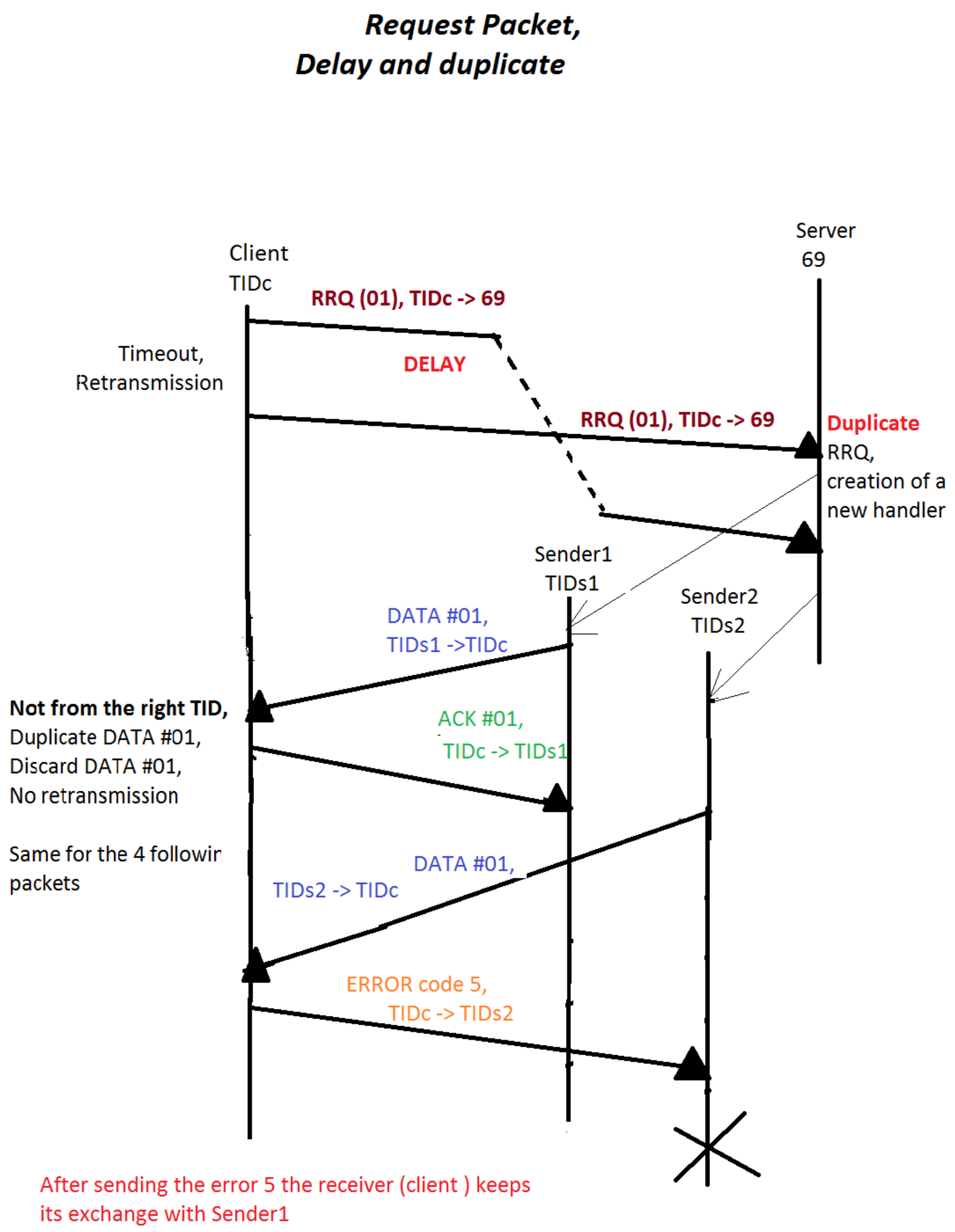


Figure 16: RRQ delay and duplicate

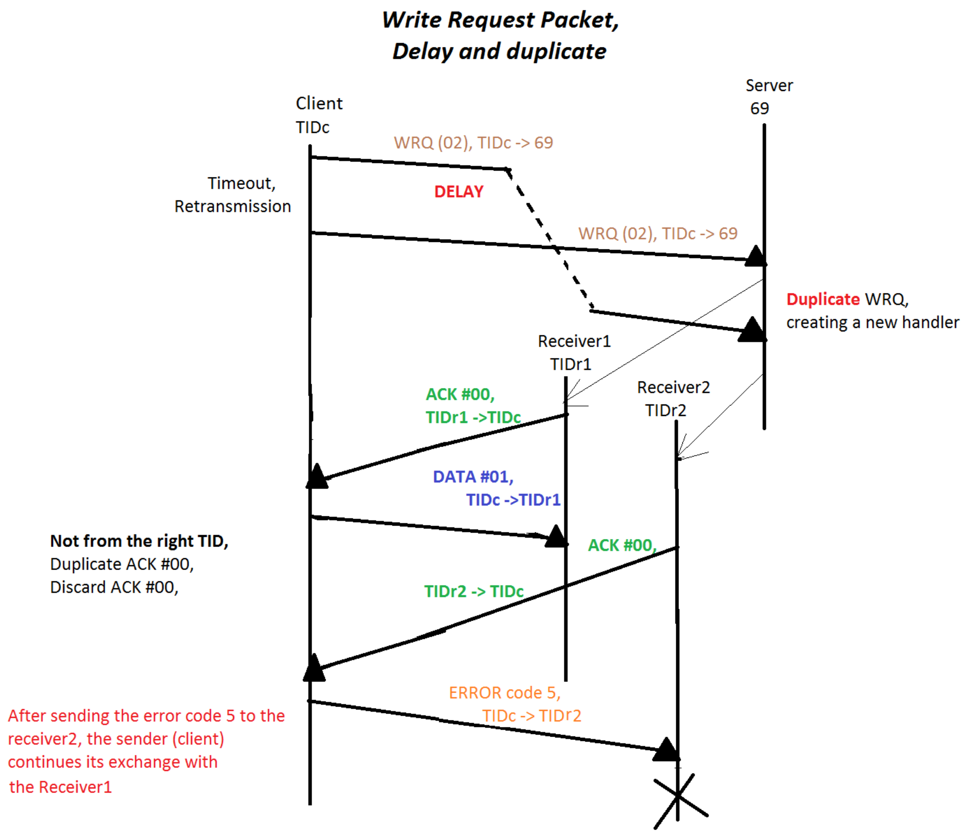
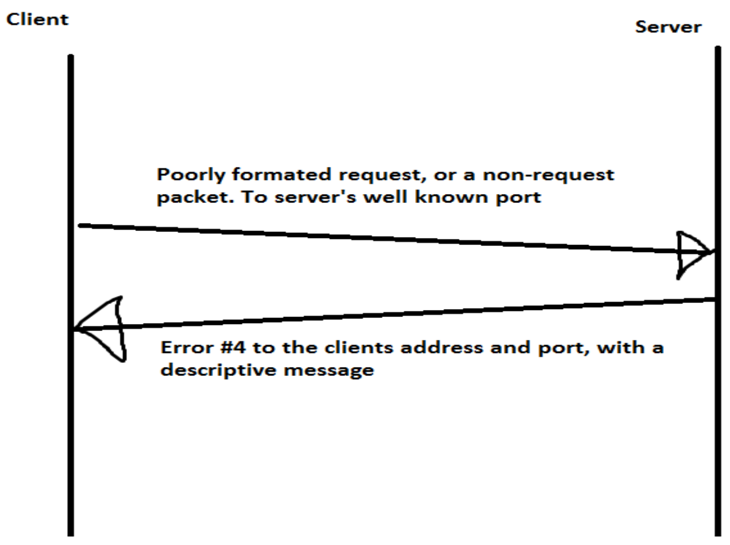


Figure 17: WRQ delay and duplicate



Client
Server
Poorly formated request, or a non-request packet. To server's well known port
Error #4 to the clients address and port, with a descriptive message

Figure 18: Bad request Error 4

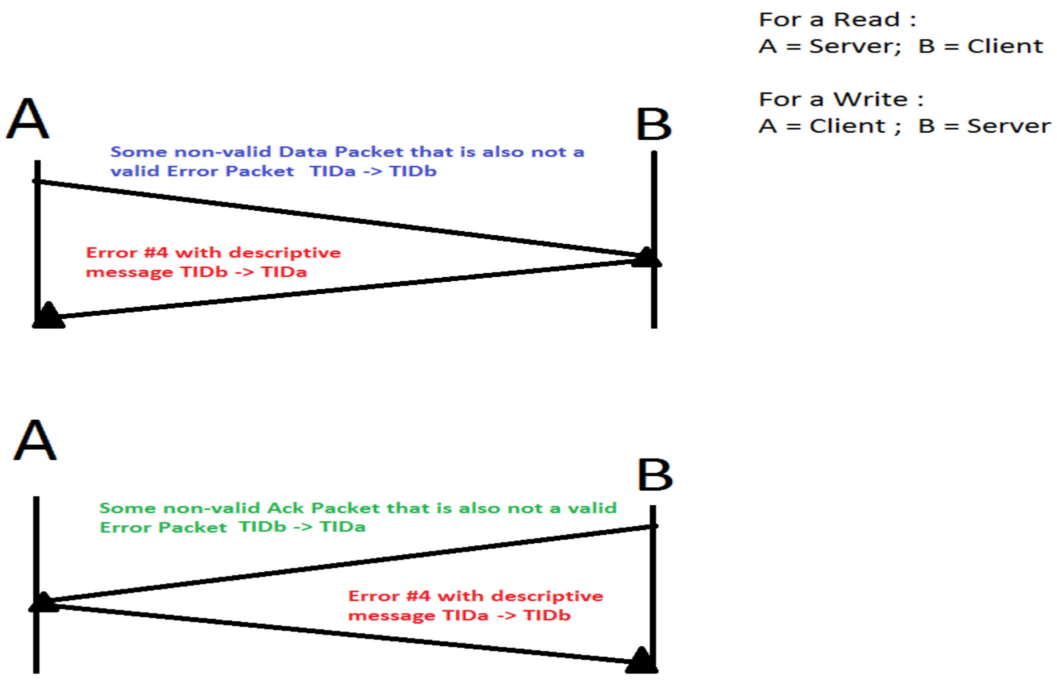

For a Read :
A = Server; B = Client
For a Write :
A = Client ; B = Server
A
B
Some non-valid Data Packet that is also not a valid Error Packet TIDa -> TIDb
Error #4 with descriptive message TIDb -> TIDa
A
B
Some non-valid Ack Packet that is also not a valid Error Packet TIDb -> TIDa
Error #4 with descriptive message TIDa -> TIDb

Figure 19: Mid-transfer Error 4

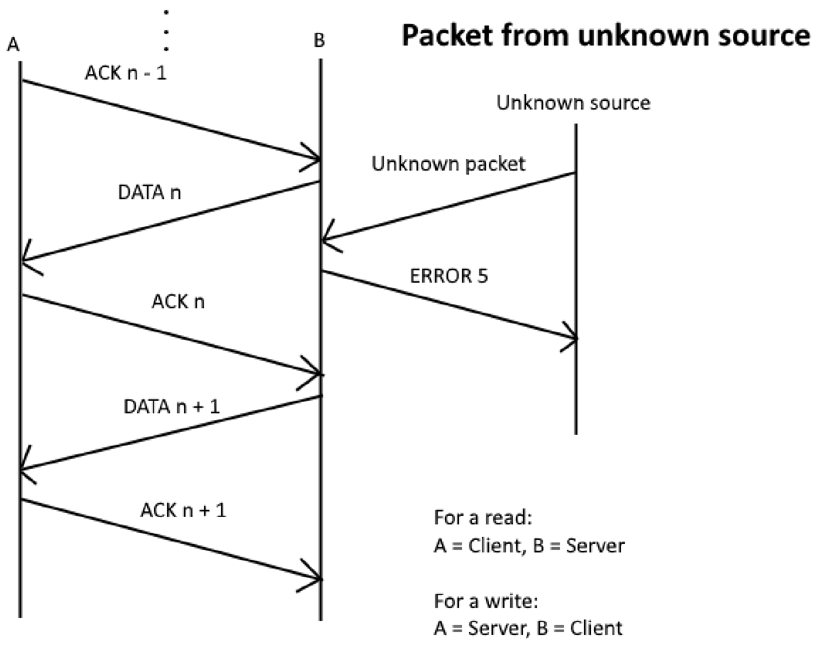


Figure 20: Packet from unknown source error 5

3 Files of the code

Our implementation is composed of 3 packages : tftp, packets and errorSim.

tftp package :
- Client.java
- ClientUI.java
- ClientController.java
- ClientErrorHandler.java
- EffectType.java
- ErrorHandler.java
- FileFactory.java
- FileType.java
- OutputHandler.java
- Receiver.java (Receiver receives the data files, writes them, and sends an acknowledgement packet when its done writing)
- Sender.java (Sender get the file, sends it to the port, receives the acknowledgement packet, and closes the file and socket)
- SendReceiveInterface.java
- Server.java
- ServerErrorHandler.java
- ServerOutputHandler.java

packets package :
- Packet.java
- DataPacket.java
- AcknowledgementPacket.java
- ReadRequestPacket.java
- WriteRequestPacket.java
- ErrorType.java
- ErrorPacket.java
- PacketFactory.java
- PacketType.java
- ReadRequestPacket.java
- WriteRequestPacket.java
- GenericPacket.java
- MistakePacket.java
- PacketsTests.java

errorSim package :
- EffectType.java
- FXCondition.java
- PacketFX.java
- IntermediateHost.java
- MakeTestFiles.java
- TestFileMaker.java

server and client directory for text files and a testFiles directory containing some test files.

Other files:
- IHErrorFile.txt (contains information to simulate loss, delay or duplicate packet, error4 and error 5)

4 Set up and Instructions

Setup Instructions :

1. Import the project into Eclipse by going to import-> Existing Projects into Workspace
2. Select the project Team17
3. Run Server.java first by right clicking and selecting "Run as Java Application"
4. Run IntermediateHost.java second by right clicking and selecting "Run as Java Application" (on the same computer than server)
5. Run Client.java third by right clicking and selecting "Run as Java Application"
6. You will see the results displayed on Client.java,IntermediateHost and Server
7. You can create test files through MakeTestFiles.java on demand directly in client or server directory by using the clientUI or ServerUI and typing 'make (filename) (number of bytes)'.

List of Commands (available on client):

1. Setting/changing server location : Type 'set server location (address)' Example : 'set server location 10.0.0.2'
2. Reading a file: Type 'read (filename)' Example: read s128.txt
3. Writing a file: Type 'write (filename)' Example: write c128.txt
4. Creating a test file :Type 'make (filename) (number of bytes)'. **Available in Client and Server**
5. Toggle Test(verbose) mode: Type 'toggle test'
6. Toggle Quiet mode: Type 'toggle quiet' **Available in Client and Server**
7. For help: Type 'help'
8. Quit the program :'quit' . **Available in Client and Server**

5 Implementation

5.1 Description

The Client provides a simple user interface that allows the user to input required values. It establishes the appropriate connection with the server and initiates the transfer. It does not terminate until the user indicates that no more files are to be transferred and it does not support concurrent file transfers.

The Error Simulation (port 23) communicates with the client and the server using DatagramSocket objects.

The ErrorSimulator uses a context-free language to read its actions towards packets, the language is described as follows:
program packetType packetNumber [args]
Available 'programs' are :

- **drop (packet) (packetNum) [noArgs]** :
  simply drop the packet, this packet vanishes

- **delay (packet) (packetNum) [timeInMillis]** :
  delay the packet for the specified time or until the condition. Other packets will be passed through while this packet is waiting

- **duplicate (packet) (packetNum) [numberOfPackets timeBetweenDuplicates]**
  every timeBetweenDuplicates (fist packet is immediate), send a packet and decrement numberOfPackets. Note if numberOfPackets=1 the packet passes unaffected

- **opcode (packet) (packetNum) [newOpcodeFirstByte newOpcodeSecondByte]**:
  change the opcode of the given packet to the new opcode (split over two arguments)

- **mode (packet) (packetNum)**
  flips a character in the fileType (octet or ascii), currently doesn't allow configuring the new fileType

- **port (packet) (packetNum)**
  creates a new socket and sends this packet from that port. User cannot specify the port to guarantee that the JVM will find an available one. Note this also drops the packet from the original sender

- **size (packet) (packetNum) [sizeModifier] {'zeroes'}**:
  lengthens or shrinks the byte array in the packet, extends if sizeModifier is positive, shrinks if negative. 'zeroes' is optionally added to extend - adds null to the end of the byte array, instead of random values.

#lines that begin with '#' are comments and are ignored, there is no multi-line commenting, or beginning a comment mid-line

Times can be replaced with conditions - conditions are met on specific packets, for example:

- **delay ack 4 cond data 7**: will delay acknowledgement packet 4 until the intermediate host sees data packet 7.

Available packets are :

- ack
- data
- readrequest
- writerequest
- error

Currently the simulator reads from [project dir]/intermediateHost/IHErrorFile.txt.
(note - for packets that do not have associated numbers - like an error packet or readrequest/writerequest, any valid number should be put in place as a placeholder)

KNOWN BUG: After passing an error packet, sometimes the intermediateHost needs to be restarted. For safety, the intermediateHost should be restarted before initiating a transfer if the previous transfer contained an error packet.

⚠**Note :** The IntermediateHost needs to be restarted if the IHErrorFile.txt is changed or when an Error Packet is received otherwise it will break the system. After altering one transfer with instructions given in IHErrorFile.txt the intermediate host just pass packets without altering them during following transfer, until an error packet is received.
Intermediate Host needs to be stopped manually (no "quit" command has been implemented so far)

The Server(port 69) consists of multiple Java threads. Thus capable of supporting multiple concurrent read and write connections with different clients. Once the server is started, it runs until it receives a shutdown command from the server operator. (toggle quiet is implemented on Server as well)

The User Interface(S) allows the user to input commands and toggle modes for example "quiet" and "verbose" whereby in the verbose mode, the client, error simulator, and server output detailed information about packets received and sent, as well as any timeouts or retransmissions of packets.

5.2 Choices

- We allow override on both side, thus no error 6 (file already exists) can be seen during a transfer between these particular server and client. However, error 6 is implemented allowing interoperability with other clients and servers that may respond in this way.
- Sender can timeout and retransmit , ignores duplicate ack (discard ack and wait for a next ack).
- Receiver can timeout but doesn't retransmit ack . For duplicate data , receiver send the ack corresponding and discard the data packet.
- When an error happened the host sends the error packet and shuts down. If the error occured before any connection (file not found or access violation for client) the client isn't shutting down but no thread is created to deal with the transfer.
  If RRQ and WRQ are lost, the client just retransmits the request.

- When sending and receiving data, different timeouts are used. The sending party retransmits after a time out, and therfor can time out multiple times. The receiving party shuts own after the first timeout, and has a timeout 5 times as long.
- The receiving party waits a full timeout after it sends the last ack, in case it got droped, or the 'last' data packet was shortened by corruption.
- If, during the last timeout, the receiver gets any packet from the sender that is not a data packet that can be explained by delay or duplication, the transfer is considered to have failed, and error 4 is sent.
- If the sender or receiver , receive a packet from an unknown TID, then the packet is discarded, an error 5 packet is created and the transfer still goes on.
  If a host receives an error 5 packet it should shut down (it means the other host doesn't accept packet from this host) .
- Finally, we allow to retransmit 5 request packet (write or read) in case packets has been dropped or delayed. If a duplicate request is received in the server , a new handler is created but shuts down after receiving an error 5 packets from the host in transfer.
- If a transfer is not complete, we decided to delete the incompleate file. This is out of conveneance, as incompleate files were littering our dirrectories, and confusing our testing. It also seemed like a amore practicle responce than leaving potentially misleading files.

The code won't be provided in hard copy because it will involved too many tees.