TFTP Implementation

Project SYSC 3303 - Real-Time concurrent System

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

December 1, 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 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.

Therefore, we decided to start our collaboration with Griffin's group code because I wrote most of it.

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 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.
- tested intermediate host.
- enhanced the interpreter to add condition.
- 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.

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.
- redrew UCM diagram to follow the current state of the implementation.
- tested everything.

Griffin Barrett(100978435) :

- wrote code to remove files if transfer incomplete.
- tested everything.

Lisa Martini (101057495) :

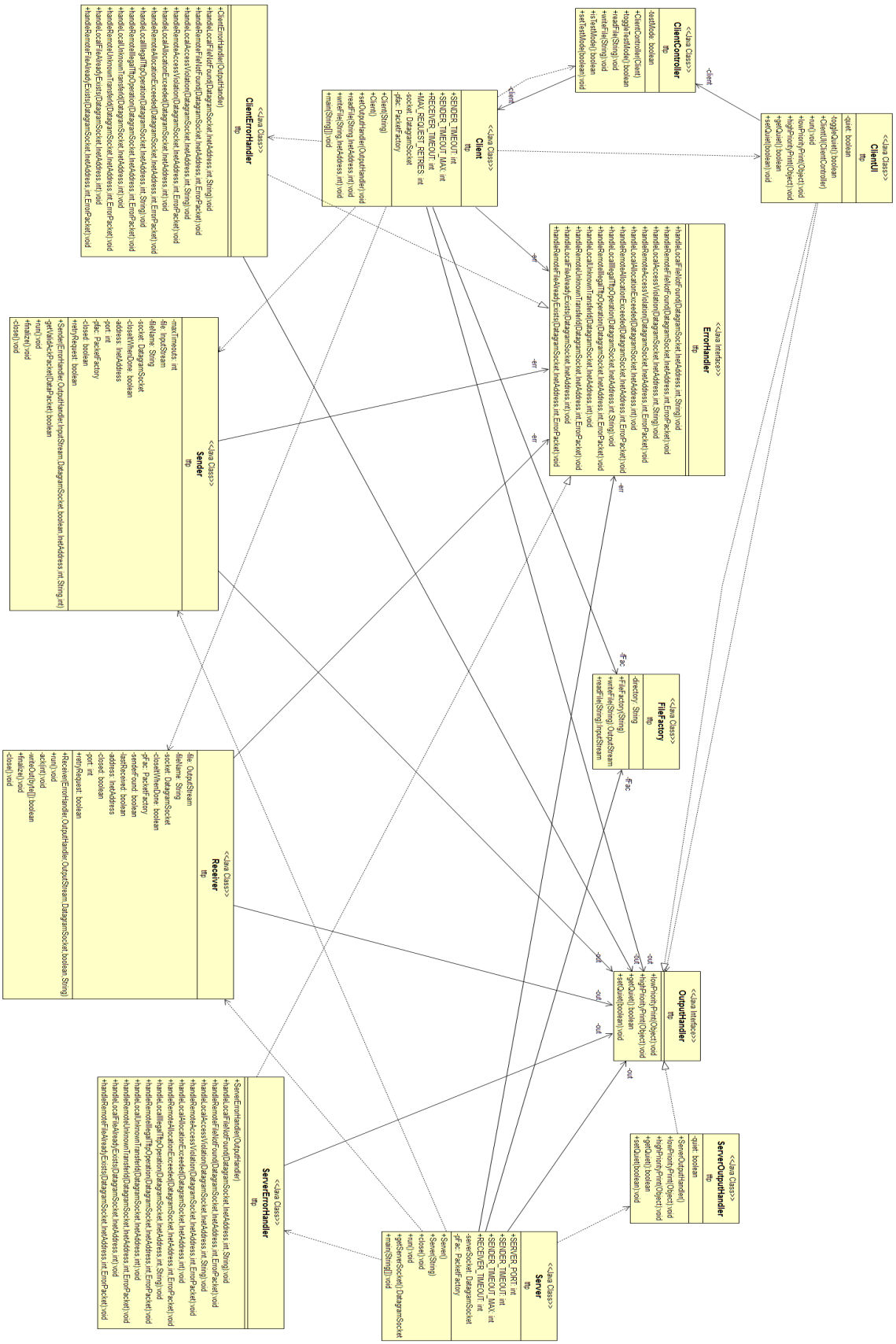
- wrote code to allow client to set/change server location to transfer a file from a different computer.
- wrote final report.
- redrew diagrams from IT2 and IT3 to follow the current state of the implementation

2 Diagrams

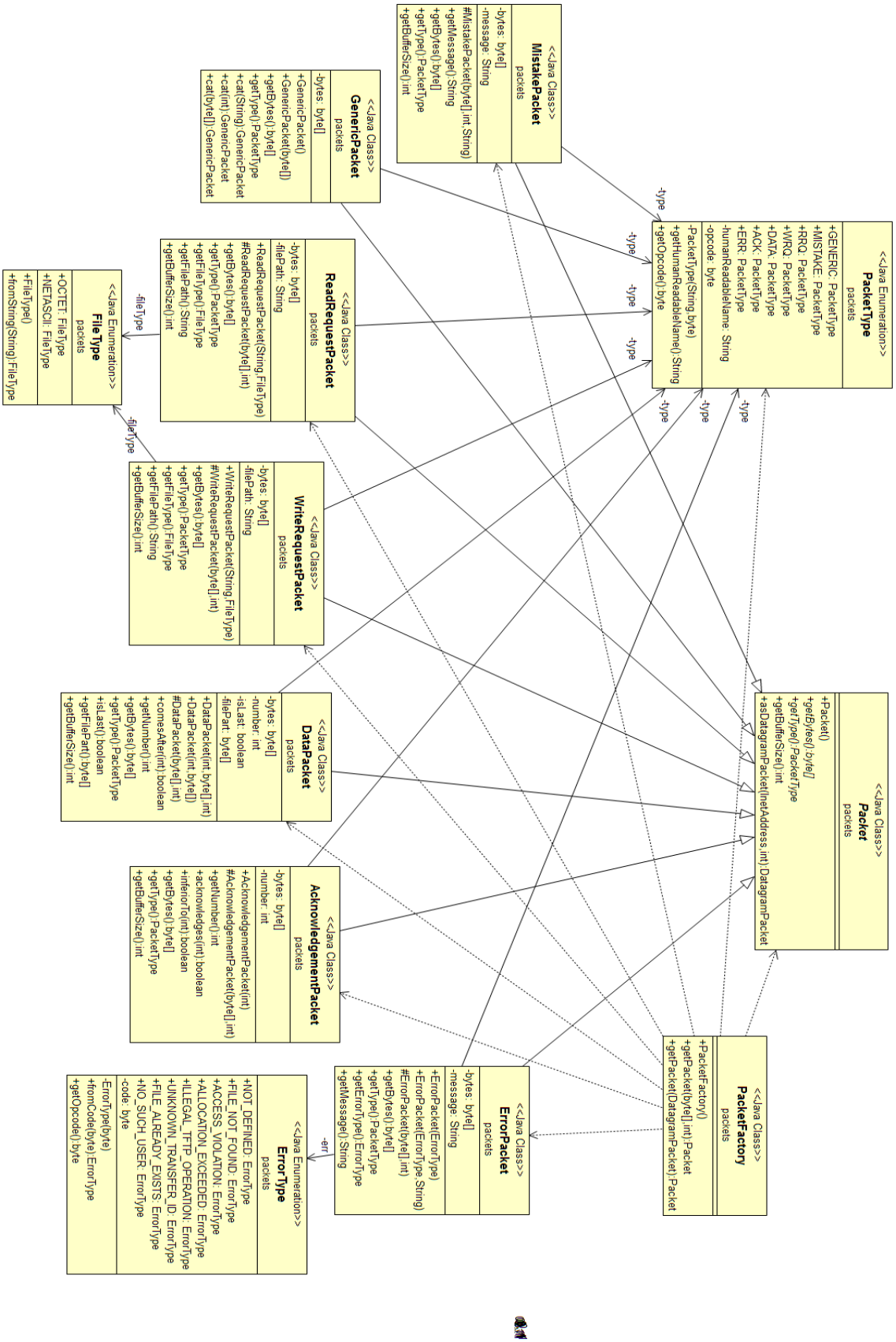
2.1 UCMs for read and write

2.2 UML Class Diagrams

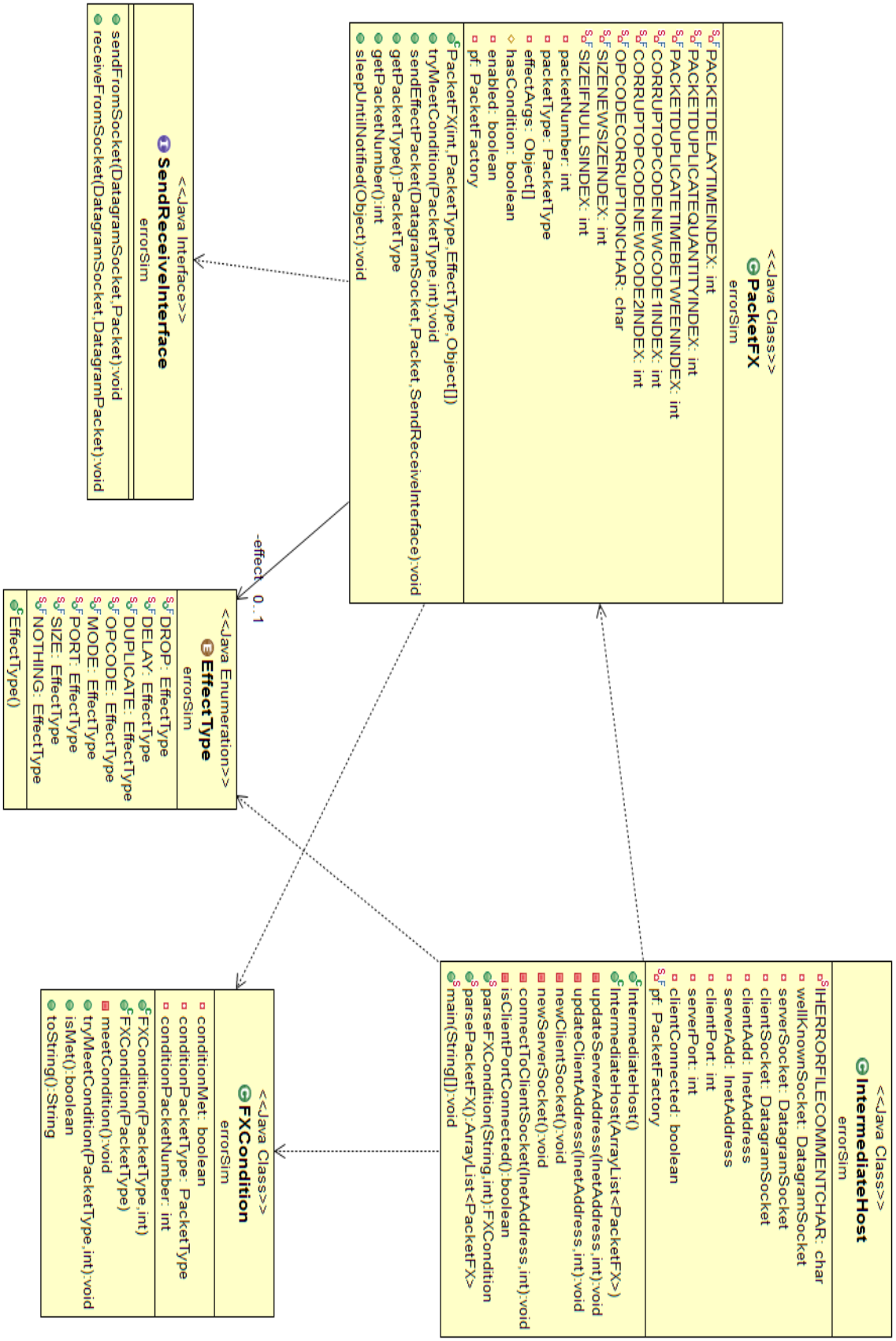
2.2.1 TFTP Package



2.2.2 Packets Package



2.2.3 ErrorSim Package



2.3 Timing Diagrams

2.3.1 I/O Errors

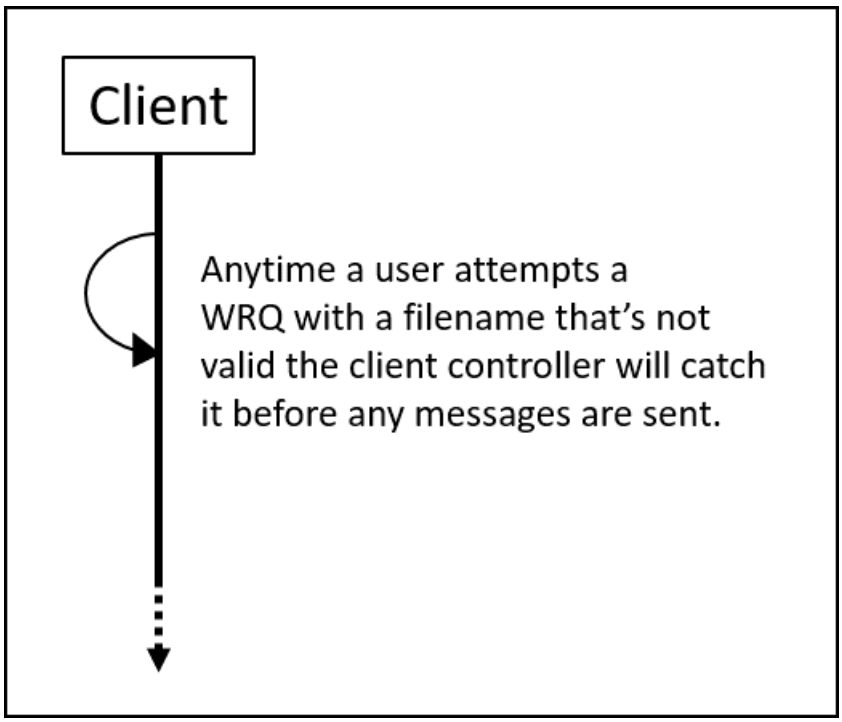


Figure 1: Client error

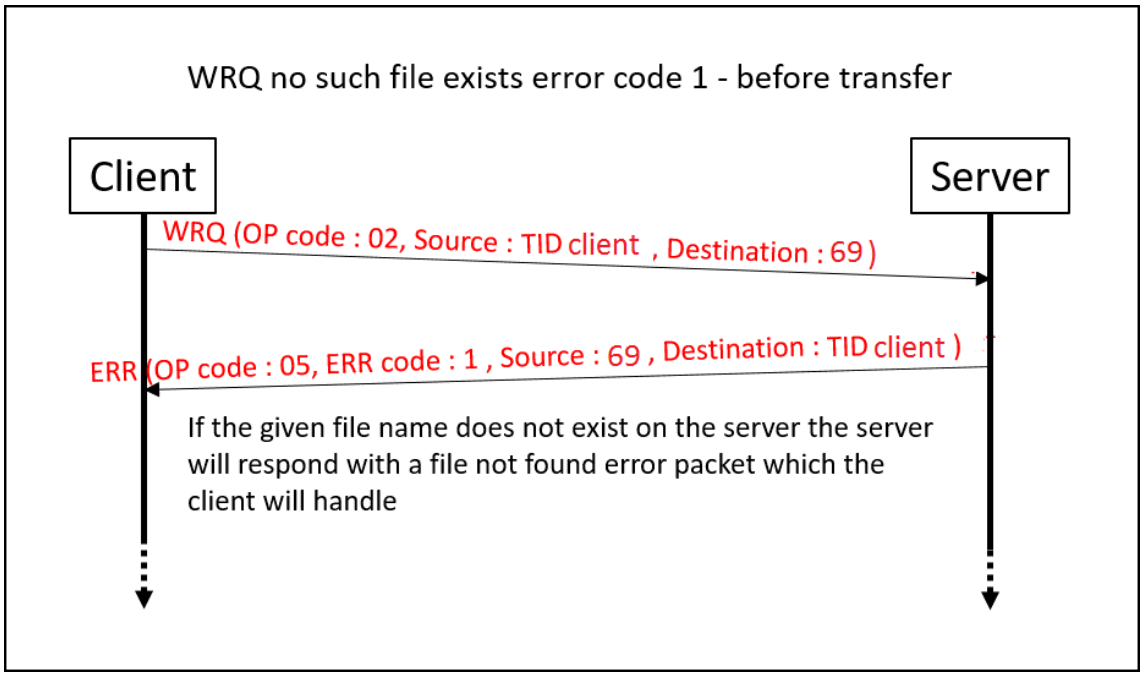


Figure 2: WRQ error code 1 : file not found

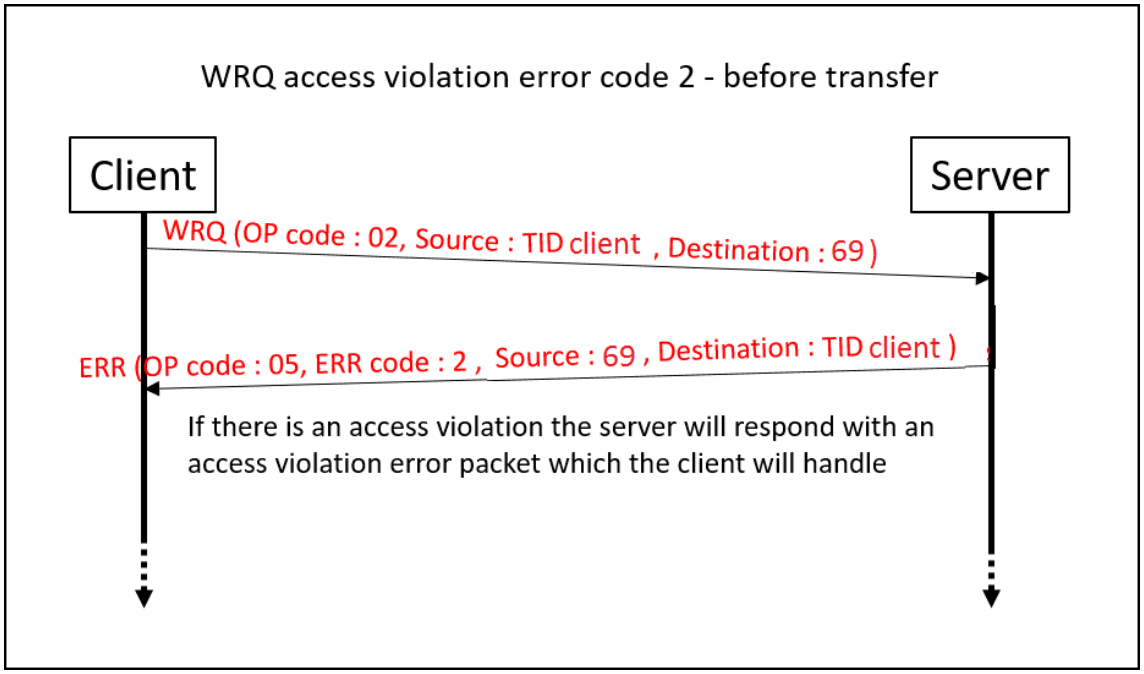


Figure 3: WRQ error code 2 : Access violation

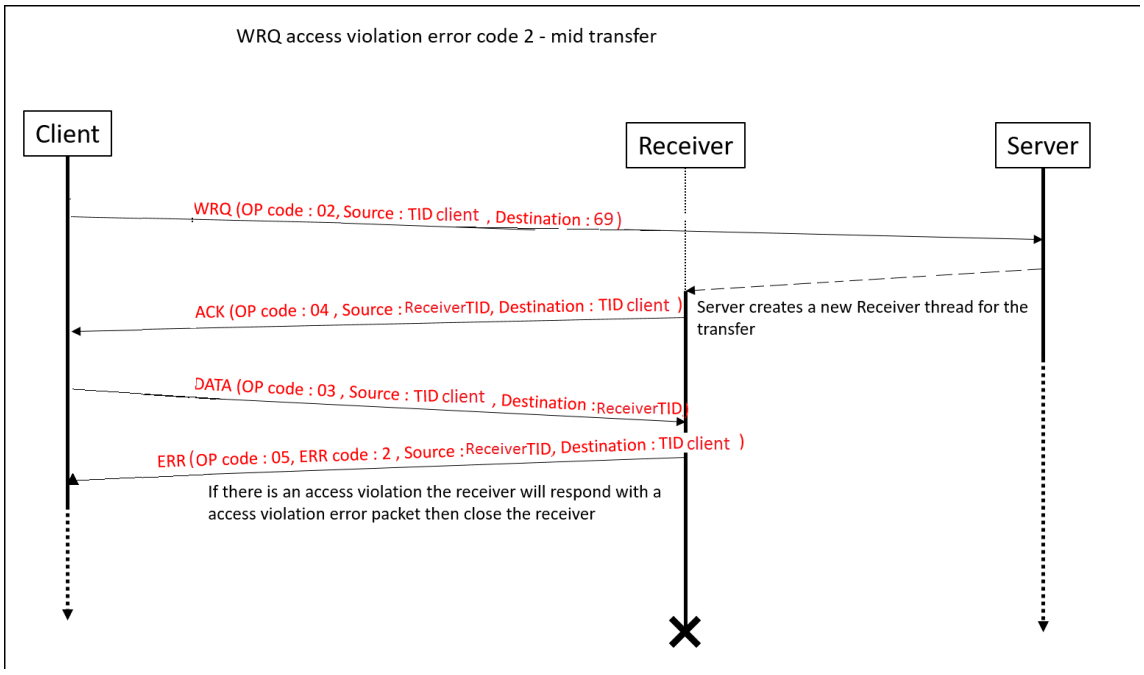


Figure 4: WRQ error code 2 : Access violation mid-transfer

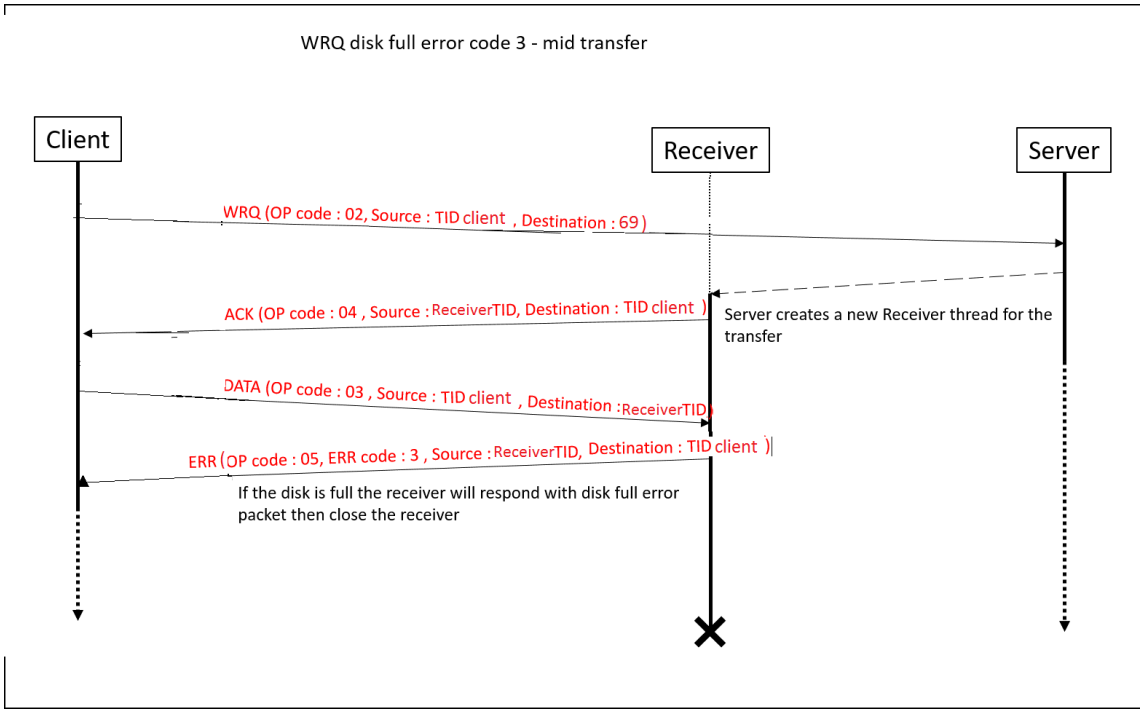


Figure 5: WRQ Error code 3, Allocation exceeded

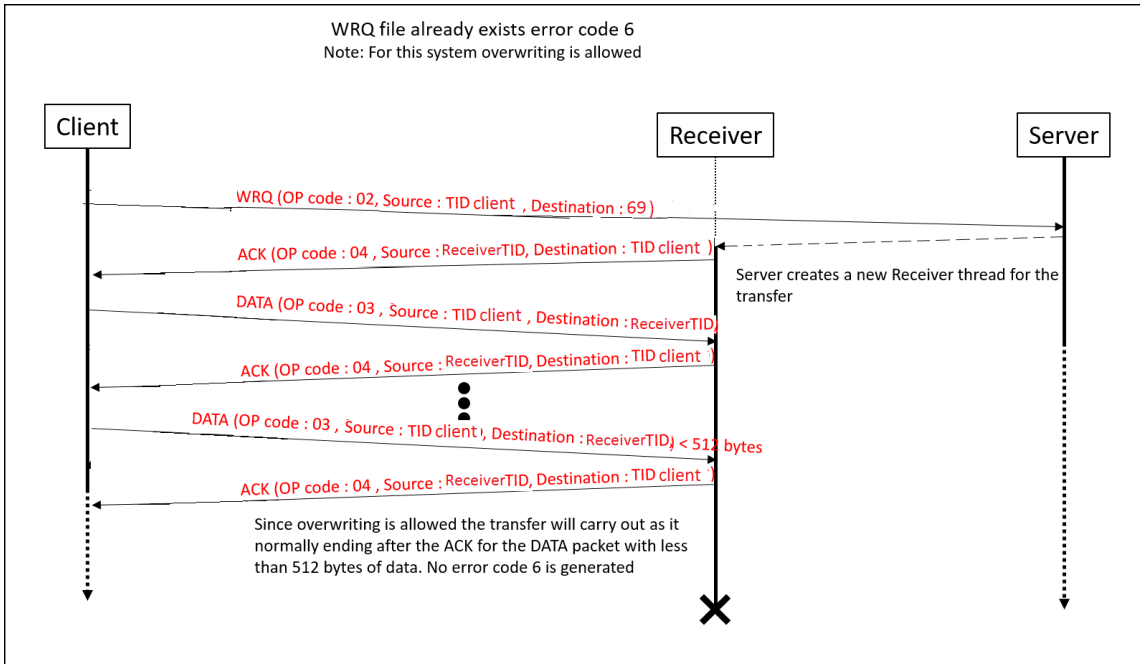


Figure 6: WRQ error code 6 : file already exists

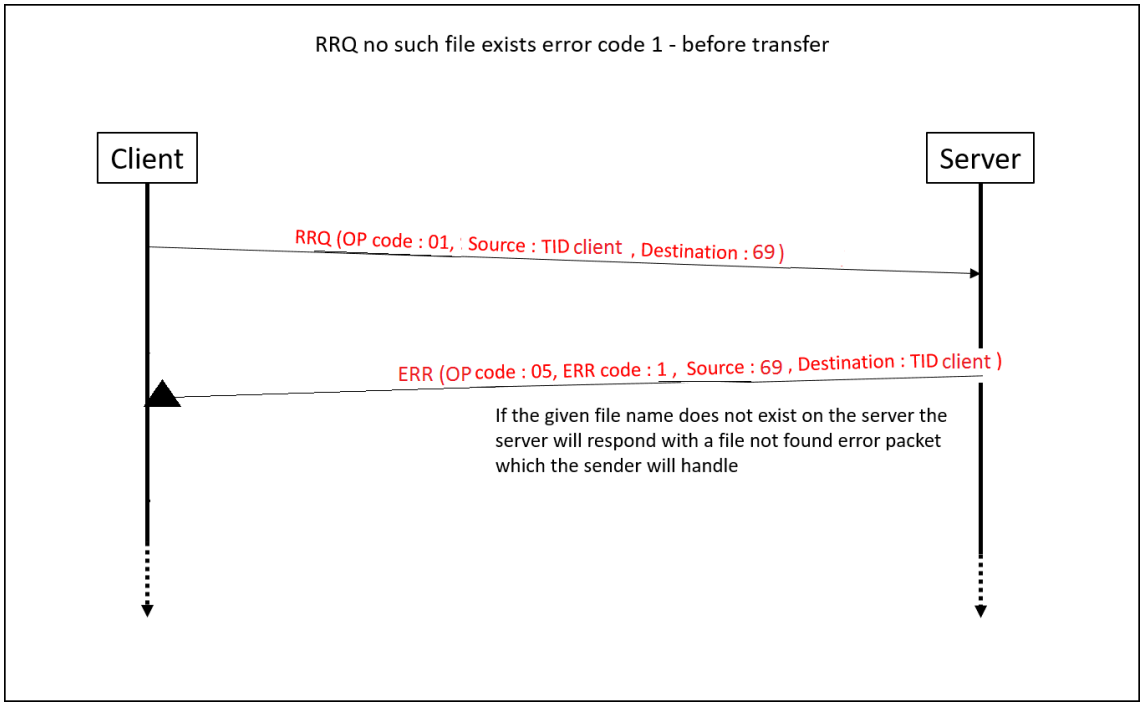


Figure 7: RRQ error code 1 : file not found

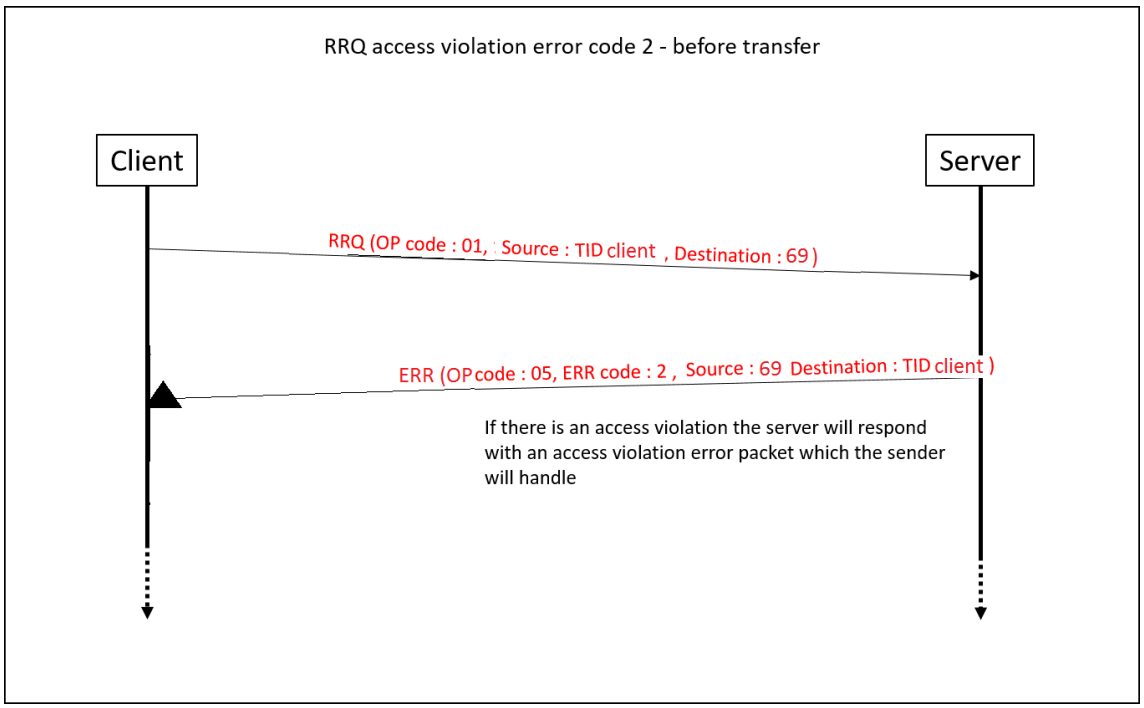


Figure 8: RRQ error code 2 : Access violation

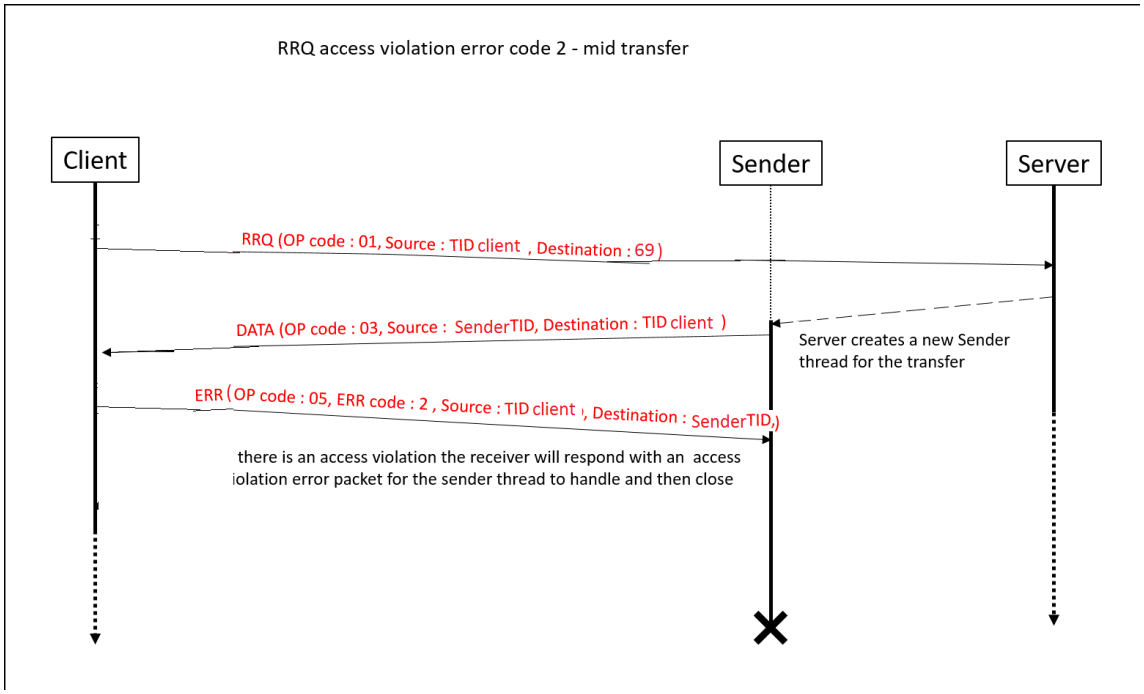


Figure 9: RRQ error code 2 : Access violation mid-transfer

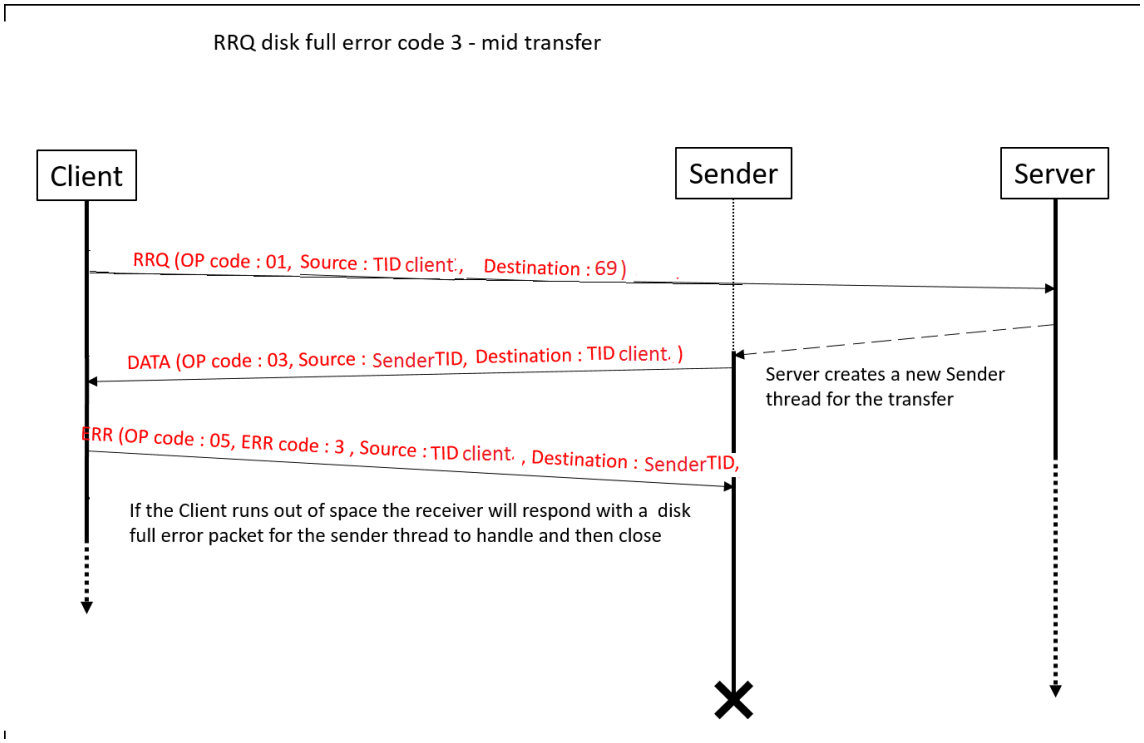


Figure 10: RRQ Error code 3, Allocation exceeded

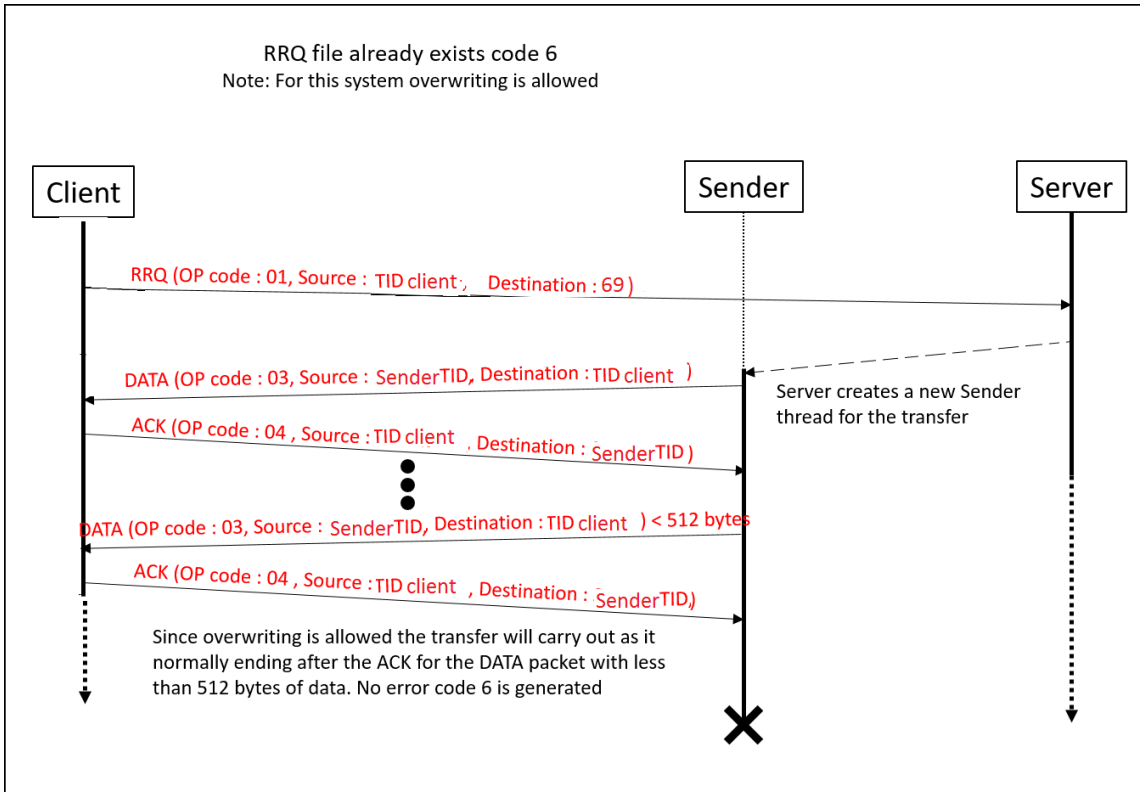


Figure 11: RRQ error code 6 : file already exists

2.3.2 Network Errors

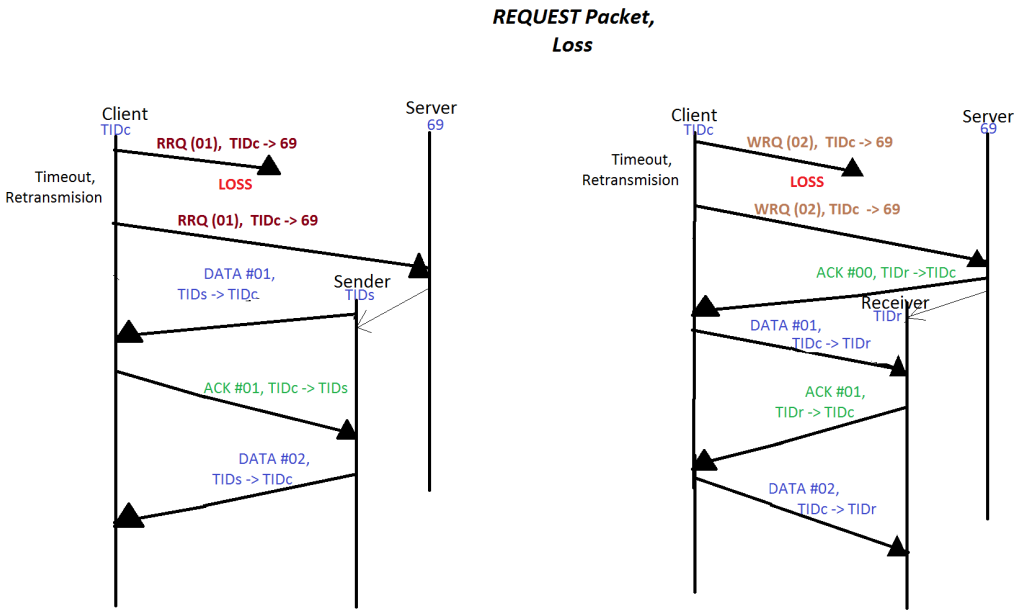


Figure 12: Request packet, loss

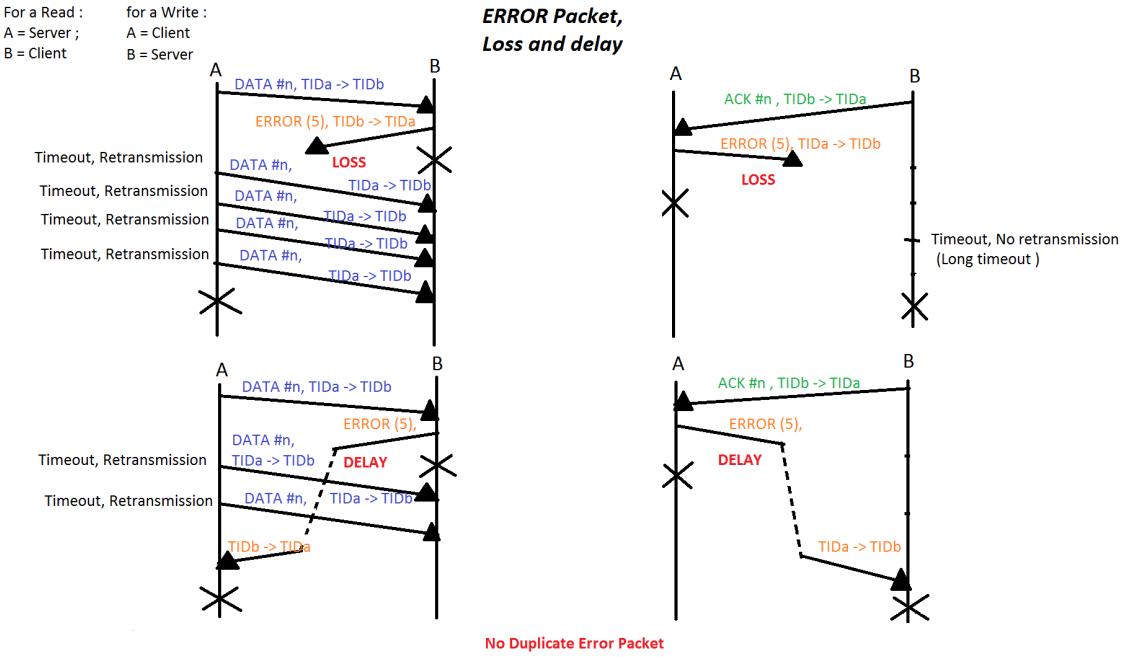


Figure 13: ERROR packet, loss and delay

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)

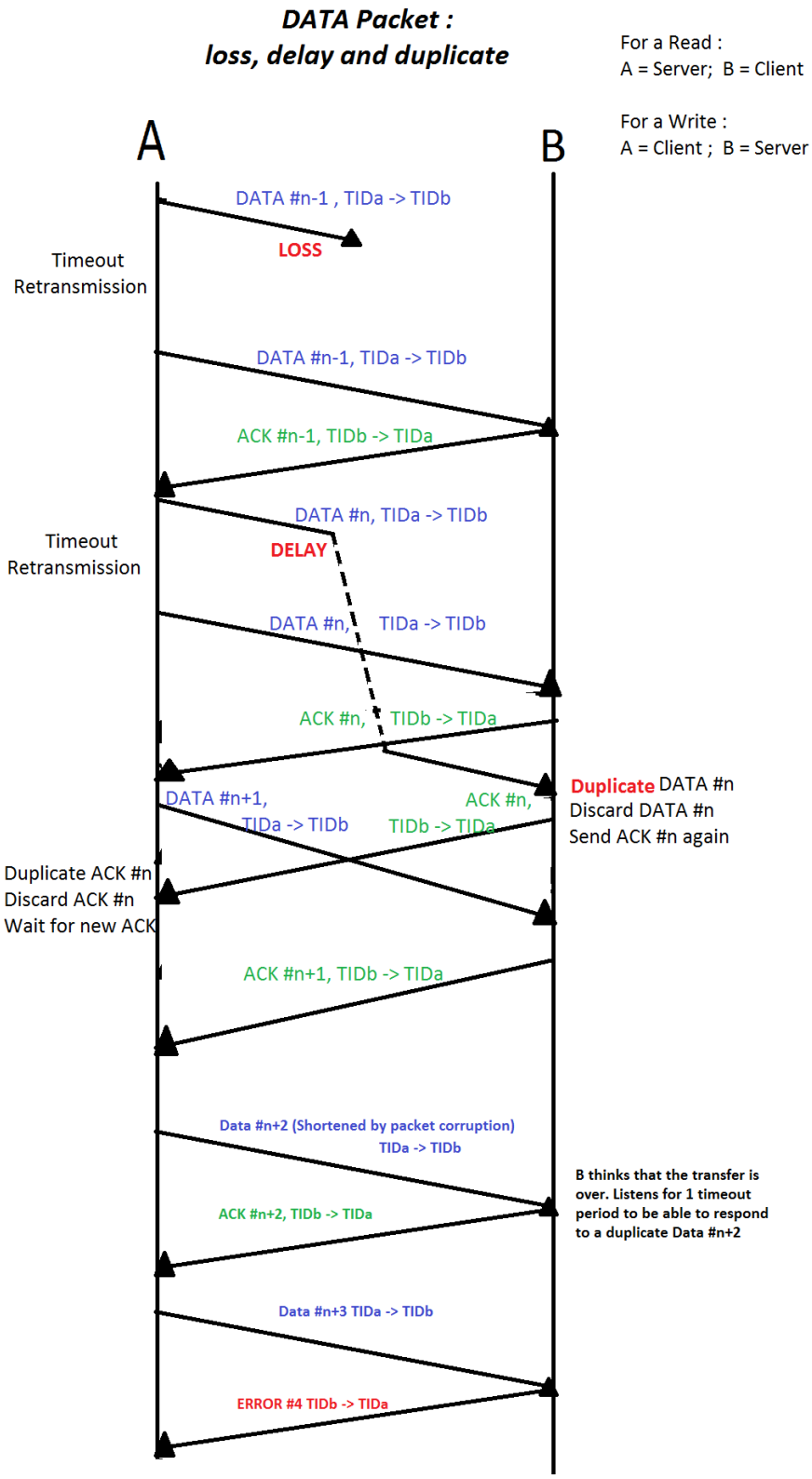


Figure 14: DATA packet, loss, delay, duplicate and shorten

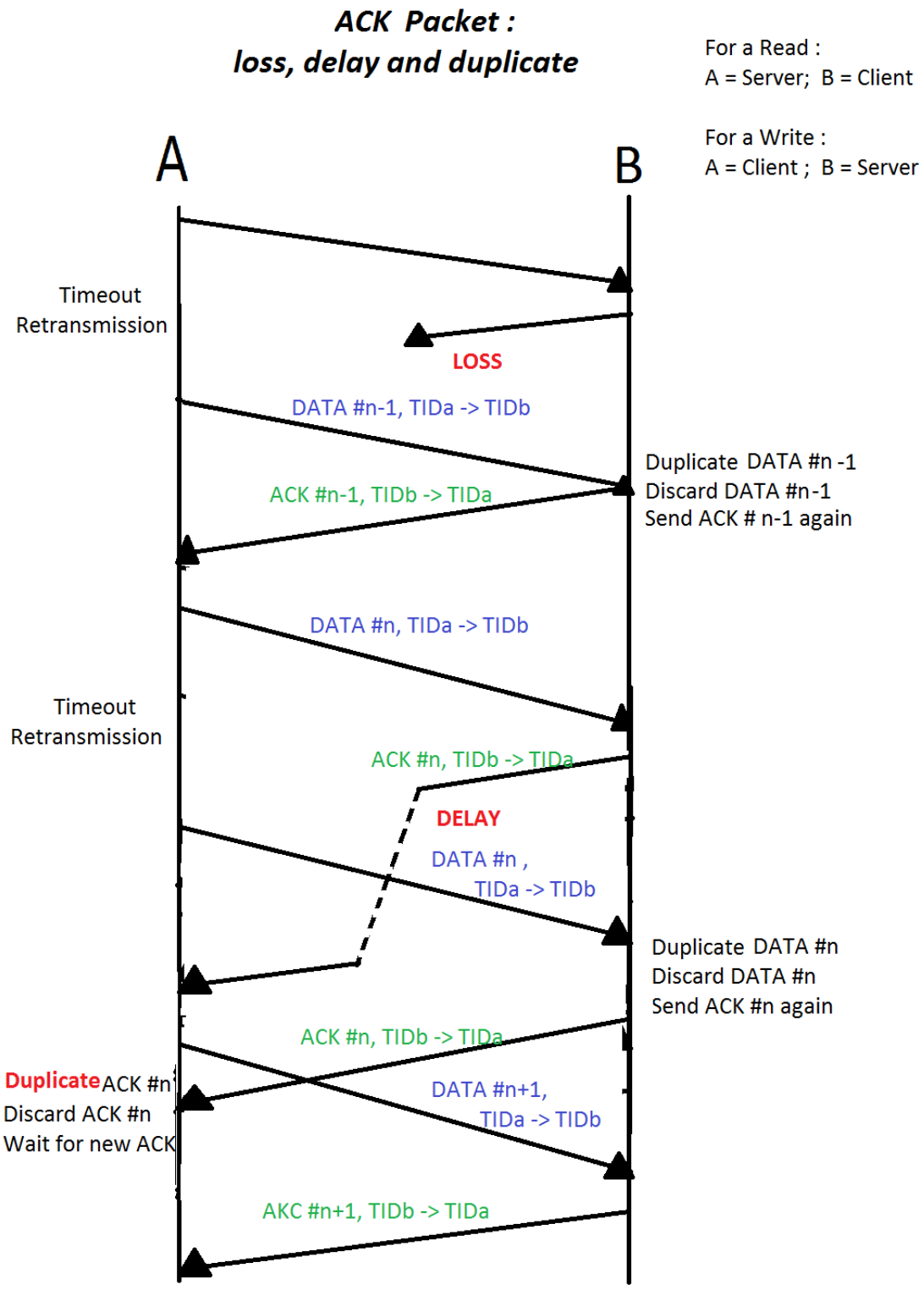


Figure 15: ACK packet, loss, delay and duplicate

2.3.3 TFTP Errors

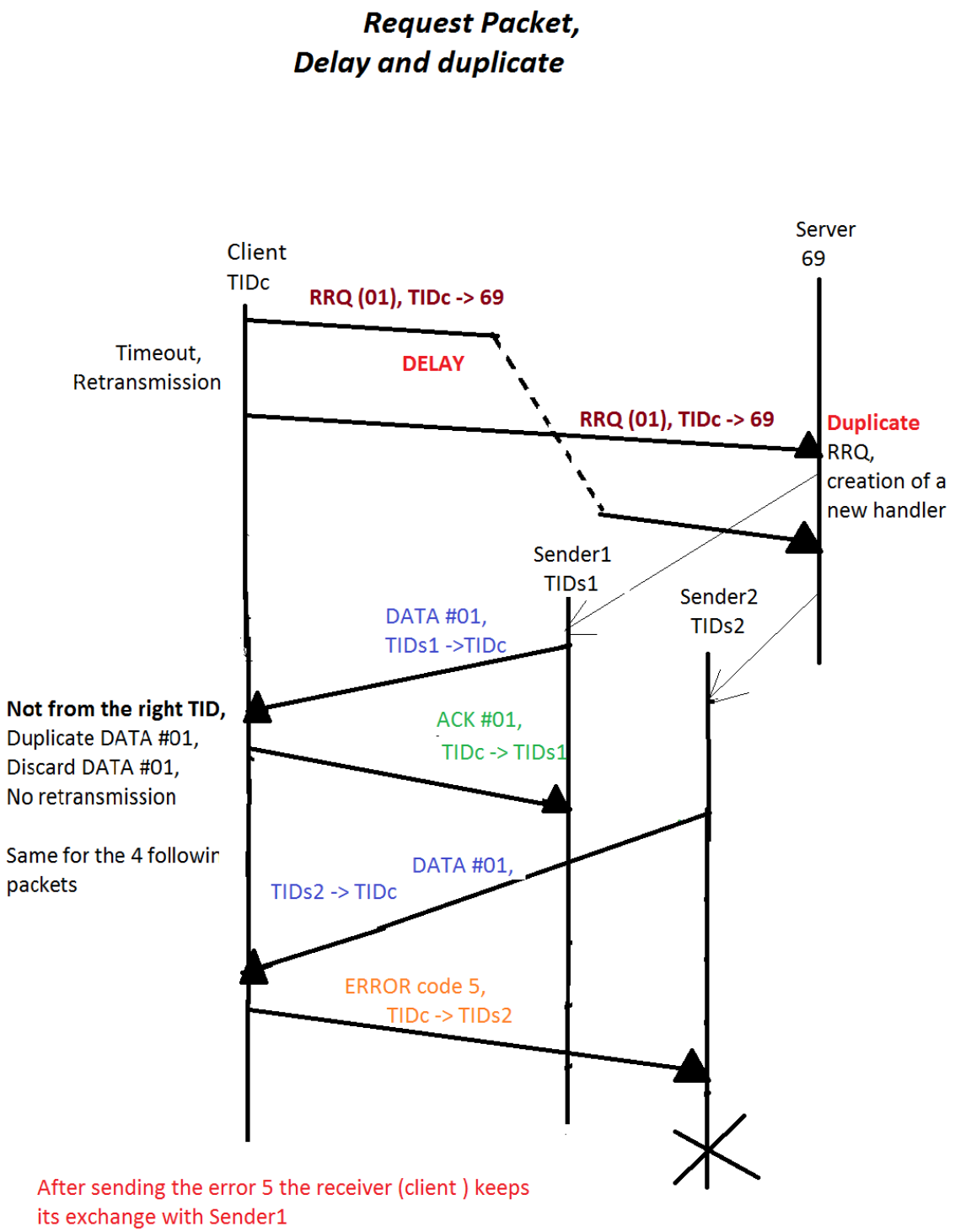


Figure 16: RRQ delay and duplicate

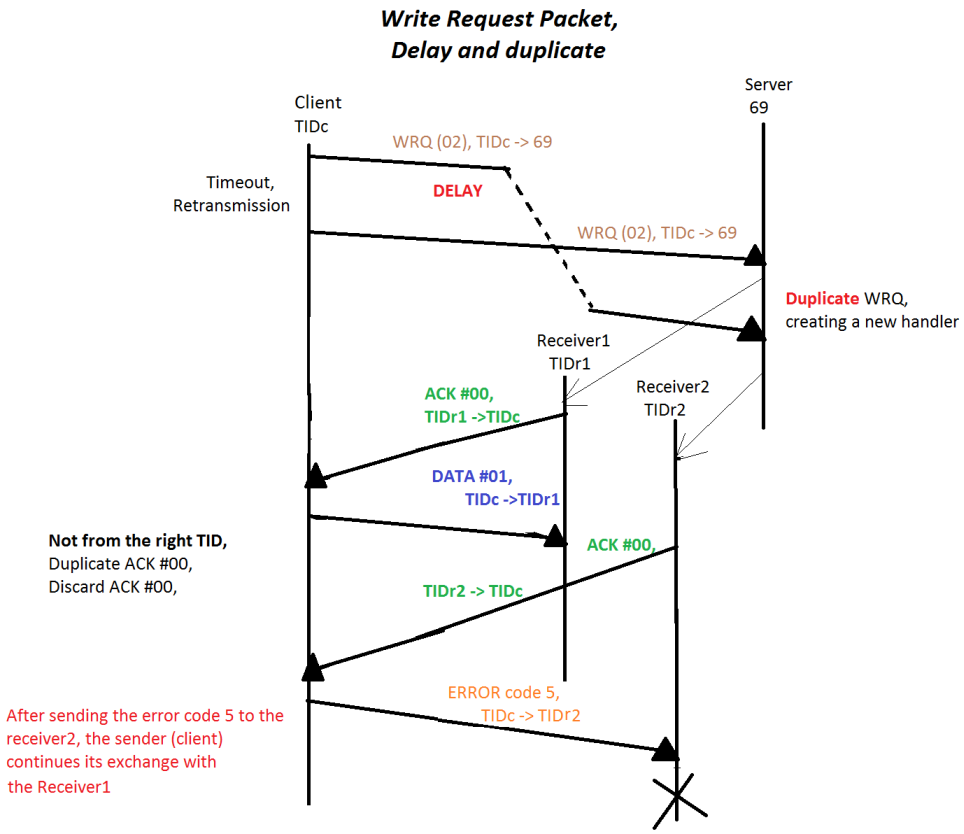


Figure 17: WRQ delay and duplicate

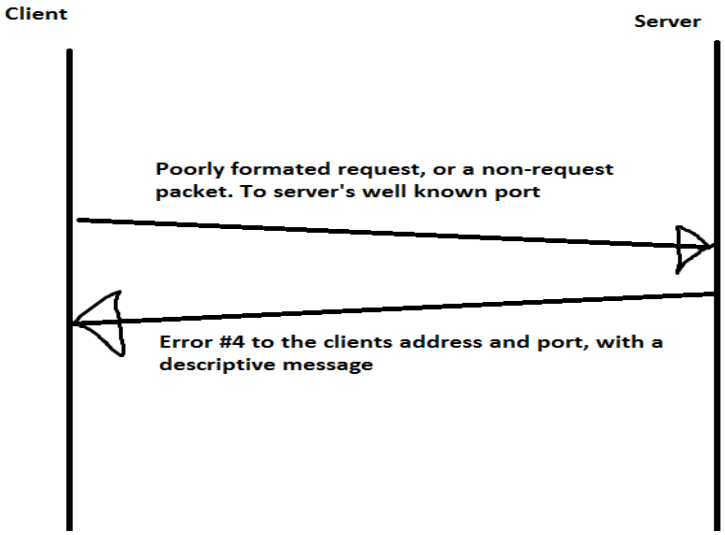


Figure 18: Bad request Error 4

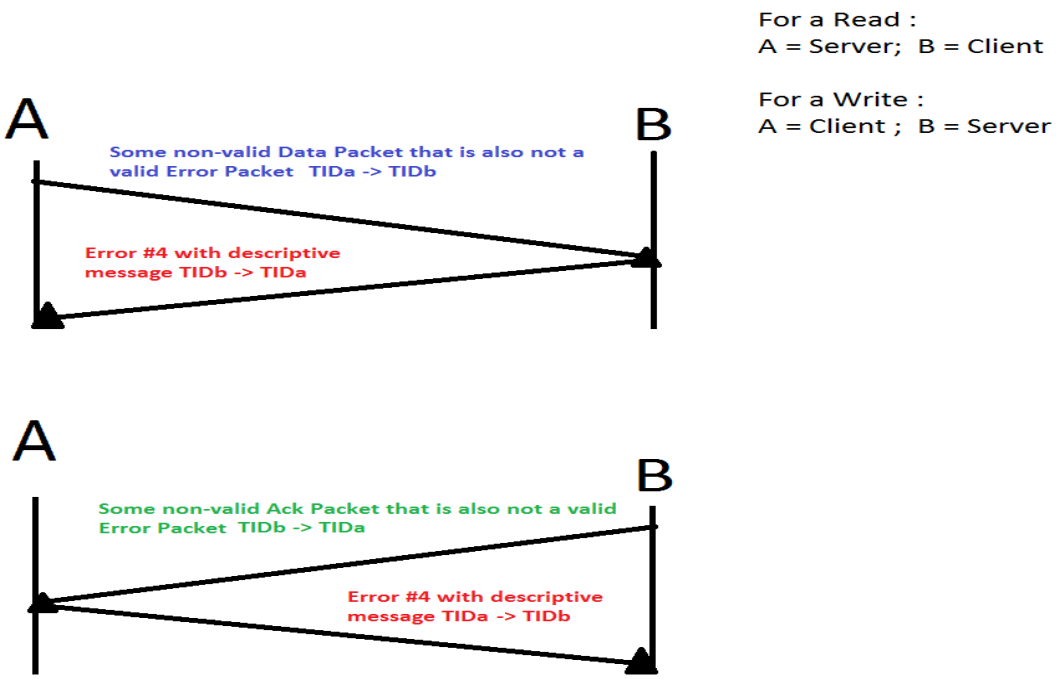


Figure 19: Mid-transfer Error 4

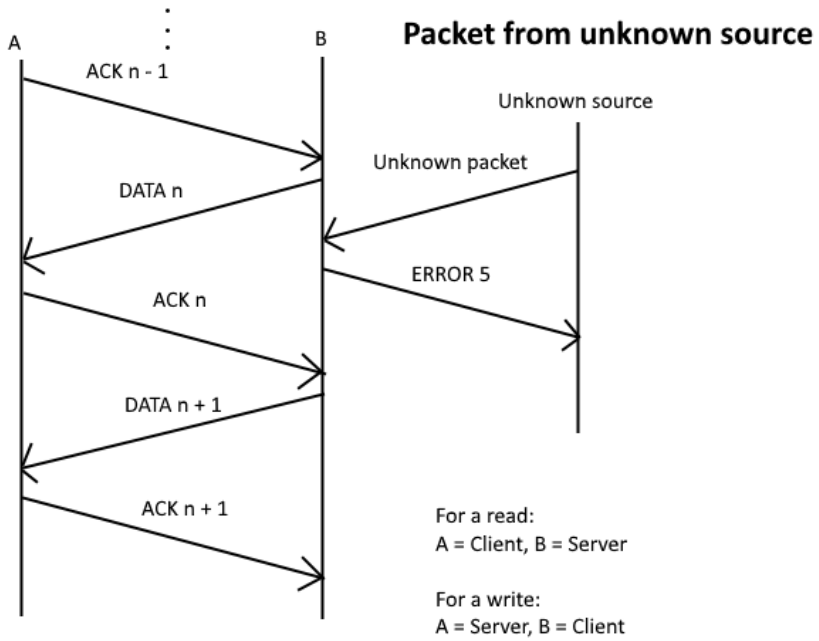


Figure 20: Packet from unknown source error 5

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

server and client directory for text 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

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. Toggle Test(verbose) mode: Type 'toggle test'
5. Toggle Quiet mode: Type 'toggle quiet' **Available in Client and Server**
6. For help: Type 'help'
7. 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 therefore by using a different client or server this error could be raised.
- 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.
- The sender and receiver have different Timeout, the receiver's is 5 times longer than sender's.

- After 5/1 timeout(s), the sender/receiver is shutting down because it means en error packet has been sent but never received.
- When last data packet is sent from the sender, the sender shuts after receiving the last ack or after 5 timeouts if error packet has been dropped.
- When last ack is sent from the receiver , the latter loops again and wait another packet in case the ack has never been received.
- The transfer is correctly complete only if the receiver times out once with the flag lastPacket received sets to true.
- Otherwise if the receiver times out then an error occured and has never been received.
- Therefore, in case a data packet is truncated by the intermediate host during the transfer, the receiver will think it is the last packet but will still be able to receive next packets until no more packets are sent.
  Hence, if a data packet is truncated the file transfer will ends properly but the file will be different.
  In the specification, No information is provided to handle a truncated data packet during a transfer. Thus,we decided to implement it this way.
- 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) .
- If a packet is corrupted an error 4 packet is created (where the corrupted packet is received) and sent to the host. Then Both host should end the transfer.
- 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 partial file, this file is overloading the system and is useless because we know that the file won't be complete.

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