# SYSC 3303 L1

Group 1

TFTP File Transfer System

Luke Newton (100999309)

Ryan Ribeiro (100997936)

Cameron Rushton (101002958)

Joe Frederick Samuel (100998314)

Kevin Sun (101000157)

# Contents

# 1 Breakdown of Responsibilities

## 1.1 Iteration 1

| | |
|---|---|
| Luke Newton | Provided a majority of the code for iteration 1 from assignment 1<br>Implemented server shutdown procedure<br>Assisted in making the server multithreaded |
| Ryan Ribeiro | Implemented steady-state file transfer/TFTP protocols between client and server |
| Cameron Rushton | Refactored the original server/client code from assignment 1 into a single program for iteration 1<br>Code reviewing/solving merging conflicts |
| Joe Frederick Samuel | Implemented steady-state file transfer/TFTP protocols between client and server<br>Created the UML class diagram |
| Kevin Sun | Provided segments of the code for iteration 1 from assignment 1<br>Implemented multithreading in the server |

## 1.2 Iteration 2

| | |
|---|---|
| Luke Newton | Fixed errors and bugs from iteration 1<br>Created all timing diagrams for iteration 2 |
| Ryan Ribeiro | Implemented a solution for error 2 (access violation)<br>Minor bug fixing + adding printouts to display error messages from received error packets |
| Cameron Rushton | Implemented a solution for error 3 (disk full/allocation exceeded)<br>Code reviewing/solving merging conflicts |
| Joe Frederick Samuel | Implemented a solution for error 6 (file already exists)<br>Updated the UML class diagram + Javadoc |
| Kevin Sun | Implemented a solution for error 1 (file not found) |

## 1.3 Iteration 3

| | |
|---|---|
| Luke Newton | Rewrote the error simulator to generate errors for testing purposes<br>Code reviewing/providing general feedback to improve solutions |
| Ryan Ribeiro | Code reviewing/providing feedback with a focus on duplicate packets<br>Updated the UML class diagram |
| Cameron Rushton | Implemented a solution to the Sorcery's Apprentice bug (This solution was modified to handle the other errors for this iteration with the help of the remaining group members)<br>Bug fixing |
| Joe Frederick Samuel | Code reviewing/providing feedback with a focus on lost and delayed packets<br>Bug fixing |
| Kevin Sun | Created all the new timing diagrams<br>Code reviewing/providing feedback with a focus on duplicate packets |

## 1.4 Iteration 4

| | |
|---|---|
| Luke Newton | Updated the error simulator to allow for the invalidation of various packets<br>Extensive testing of the different type of the different types of invalidated packets<br>Bug fixing |
| Ryan Ribeiro | Implemented a solution for error 5 (unknown transfer ID)<br>Testing of code in preparation for the first demo<br>Updated the UML class diagram |
| Cameron Rushton | Implemented a solution or error 4 (illegal TFTP operation) |
| Joe Frederick Samuel | Updated the menu in error simulator<br>Code reviewing/minor bug fixing |
| Kevin Sun | Code reviewing/minor bug fixing |

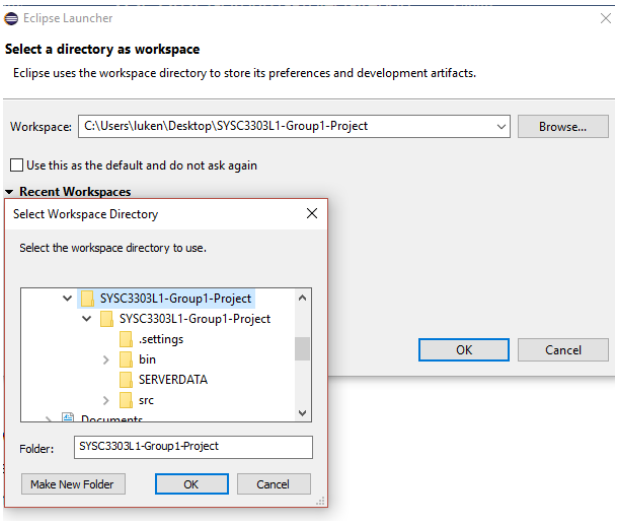
## 1.5 Iteration 5

| | |
|---|---|
| Luke Newton | Modified the program to allow for file transfers between two different<br>Testing of code in preparation for the final demo<br>Worked on the final report (setup instructions/testing/diagrams)<br>Code review of the final product |
| Ryan Ribeiro | Bug fixing<br>Testing of code in preparation for the final demo<br>Worked on the final report<br>Updated the final UML class diagram (responsibility breakdown/diagrams)<br>Code review of the final product |
| Cameron Rushton | Code review of the final product/Final testing |
| Joe Frederick Samuel | Code review of the final product/Final testing |
| Kevin Sun | Code review of the final product/Final testing |

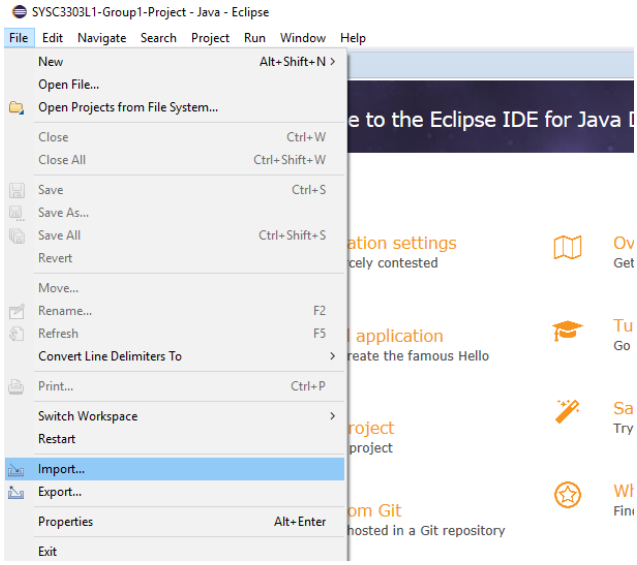
# 2 Set Up Instructions

The zip folder provided alongside this report is the eclipse project files and source code. After downloading this zip file, extract it to any location on your machine.

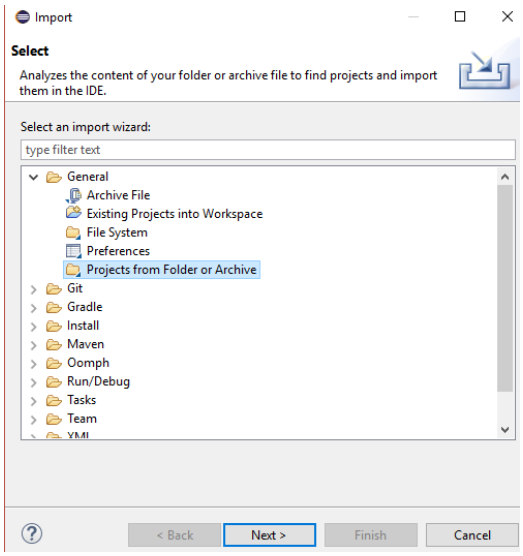
Next, open Eclipse and use this folder just extracted as the workspace:



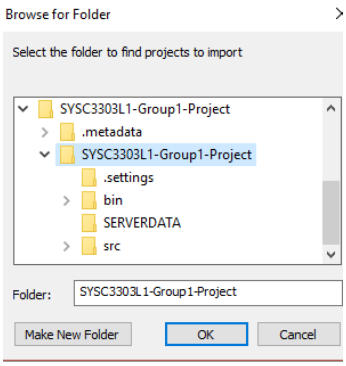

In the top right corner of Eclipse, select "File->Import...":



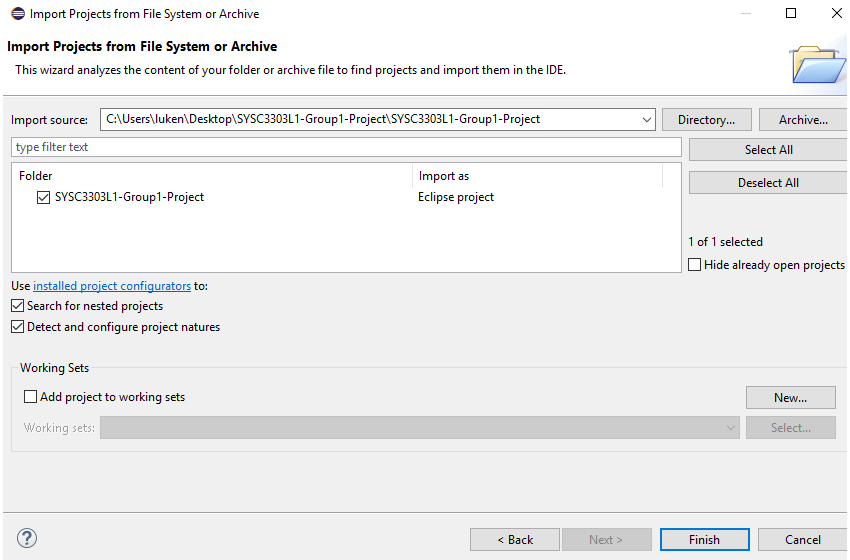

Select "General->Projects From Folder or Archive" and click "Next >":



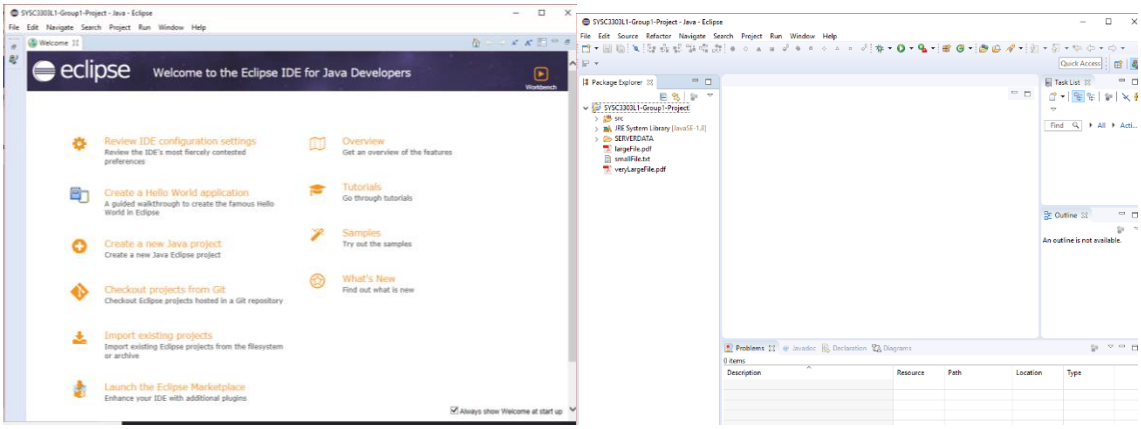

In this window, enter into the Import source textbox the path to the eclipse project in the workspace folder, or navigate to it through the window displayed by clicking "Directory...":



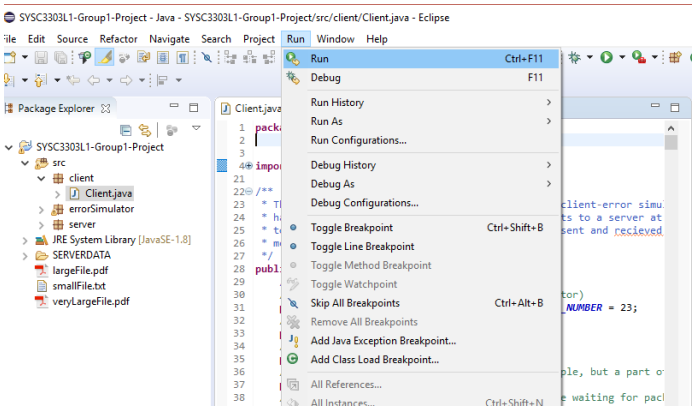

After navigating to the eclipse project folder, the display should look similar to the image below. Select "Finish":



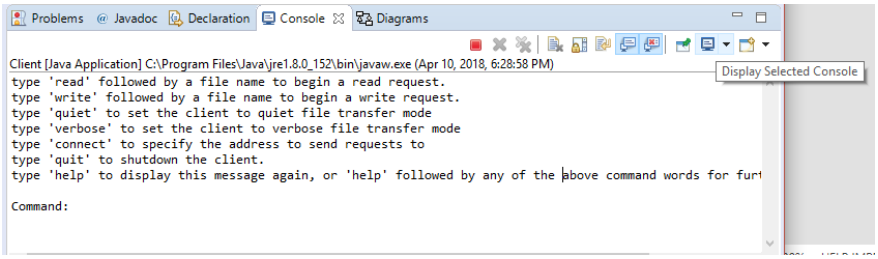
The main eclipse window should now be open, close the welcome tab and the project should be visible in the left-hand side Package Explorer:



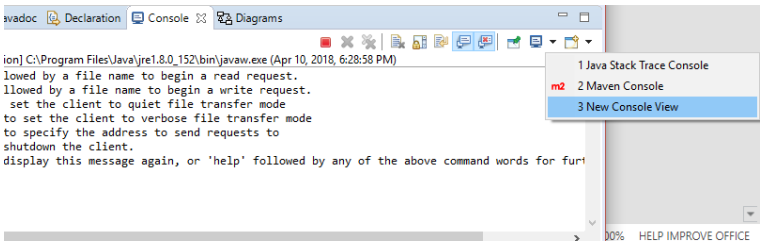

The src folder contains all the source code for the project. Double clicking on any .java file will open it in the main window. Double click on Client.java in the client package to open it, go to the toolbar and select "Run->Run" to run the client program. Follow the same steps to run Server.java in the server package to run the server, and ErrorSimulator.java in the errorSimulator package the run the error simulator (Note: if the client and server are being run on separate machines, ensure that the error simulator is on the same machine as the server):




A console display the programs' UI will have opened at the bottom of the screen. To cycle through the programs' consoles, click the "Display Selected Console" button, or click the drop down besides it to select which console to see:




To create multiple consoles to view multiple parts at once, use the "Open Console" button in the top right of the console and select "3 New Console View":




Ensure that any files that the server should have access to are located in the SERVERDATA folder on the machine the server is running on. Place files the client should have access to in the project folder (the same folder level containing SERVERDATA and src).

# 3 Test Instructions

All three of the client, server, and error simulator display information to the console about each incoming packet in the following format:

Host: IP address followed by port number

Message Length: number of bytes in the data portion of the packet

Type: the operation code of the packet (RRQ, WRQ, DATA, ACK, or ERROR)

Containing: the data in the packet as a string (useful while transferring .txt files)

Contents as raw data: the data in the packet as a byte array

In addition, when the client and server either timeout or encounter an error, an error message is displayed on the console.

Both the Client and Server have timeouts associated with their message receiving which are set to 5000 milliseconds. If testing would require longer or shorter timeouts, the constant TIMEOUT_MILLISECONDS must be changed in the Client.java and Server.java files.

## 3.1 Client

Files which "belong" to the client (meaning the simulated file space of the client) are located in the top-level project folder. This is the default space of where files are read/written in java, however a full pathname can be specified to write files elsewhere on the machine to the server.

The Client has a text-based UI which allows for interaction and testing of the system.

To perform a read request, type into the client "read" followed by the filename you which to read from the server. Note that this filename must have no whitespace and include the file extension. After a read request is completed, the client prompts the user to enter a filename/path to save the read file as, which is stored in the default client file space described above, or at the location specified by the provided path.

To perform a write request, type into the client "write" followed by the filename/filepath to be written to the server. As with read requests, this name cannot contain whitespace and must include the file extension. Once the write request is completed, the file can be found in the 'SERVERDATA' folder in the project folder

The client can be toggled between a "quiet" and "verbose" mode by entering either of the keywords into the client. This determines whether each packet's data is printed to the console during file transfer (verbose), or if the display remains empty (quiet mode). The client is by default set to verbose; a key difference between the two is that while quiet mode does not give debugging info, it is much faster.

The client can be connected to servers running on the local machine or any other machine the local machine can connect to. By default, the client is set to connect to a server on the local host, but this can be changed by typing "connect" followed by the IP address of the machine to connect to. To return to the local connection type "connect local host". An important note here is that testing on lab machines

requires the user to enter the number on the front of the machine (ex. Cb5109-<computer#>), and there were unresolvable issues with connecting non-lab machines to lab machines.

The client can be closed by typing "quit" into the console.

This list of commands can be viewed in the program by typing "help" to display a list of all commands, or "help" followed by any command keyword to get more information.

## 3.2 Error Simulator

The error simulator has a text-based UI for the purpose of creating artificial errors to disrupt normal read or write requests.

By default, the error simulator is set to normal mode, where no artificial errors are added. To return to this mode at any time, type "normal" into the console. It is advisable to do this in between each different error being tested to ensure the system does not try to create multiple errors at once, which it was not specifically designed to do.

To duplicate a packet, use the "duplicate" keyword followed by the type of packet to duplicate. If duplicating a read or write request type "duplicate" followed by RRQ or WRQ and then the number of milliseconds until the packet is to be duplicated (i.e. duplicate <WRQ/RRQ> <time until packet duplicated>). If duplicating a data or acknowledge packet, the block number must also be specified (i.e. duplicate <DATA/ACK> <block#> <time until packet duplicated>).

To drop a packet, use the "lose" keyword followed by the type of packet to drop. If duplicating a read or write request type "lose" followed by RRQ or WRQ (i.e. Lose <WRQ/RRQ>). If dropping a data or acknowledge packet, the block number must also be specified (i.e. Lose <DATA/ACK> <block#>).

To delay a packet, use the "delay " keyword followed by the type of packet to delay. If delaying a read or write request type "delay " followed by RRQ or WRQ and then the number of milliseconds the packet is to be delayed for (i.e. delay <WRQ/RRQ> <time packet delayed for>). If delaying a data or acknowledge packet, the block number must also be specified (i.e. delay <DATA/ACK> <block#> <time packet delayed for>).

To invalidate the data within a packet, use the keyword "invalid" followed by the type of invalidation and the packet to invalidate the data on. If invalidating data on a read or write request, type "invalid" followed by the choice to invalid mode, filename, opcode, or TID, and either RRQ or WRQ (i.e. invalid <mode/filename/opcode/TID> <RRQ/WRQ>). For data and acknowledge packets, only opcode and TID can be altered, but the block number of the packet to invalidate must be supplied (i.e. Invalid <opcode/TID> <DATA/ACK> <block#>).

The error simulator can be closed by typing "quit" into the console.

This list of commands can be viewed in the program by typing "help" to display a list of all commands, or "help" followed by any command keyword to get more information.

## 3.3 Server

Files which "belong" to the server (meaning the simulated file space of the server) are located in a folder called 'SERVERDATA' that exists in the project folder. Files can only be read from this folder by the client, and all files written to the server end up here.

The server can be closed via user input of a 'quit' command. This command will not immediately quit; the server will block any new requests and wait until all received requests have been completed before it shuts down. Beyond this and the above debugging information, the server does not feature other testing features.

# 4 System Diagrams

## 4.1 UML Class Diagram

## 4.2 Timing Diagrams

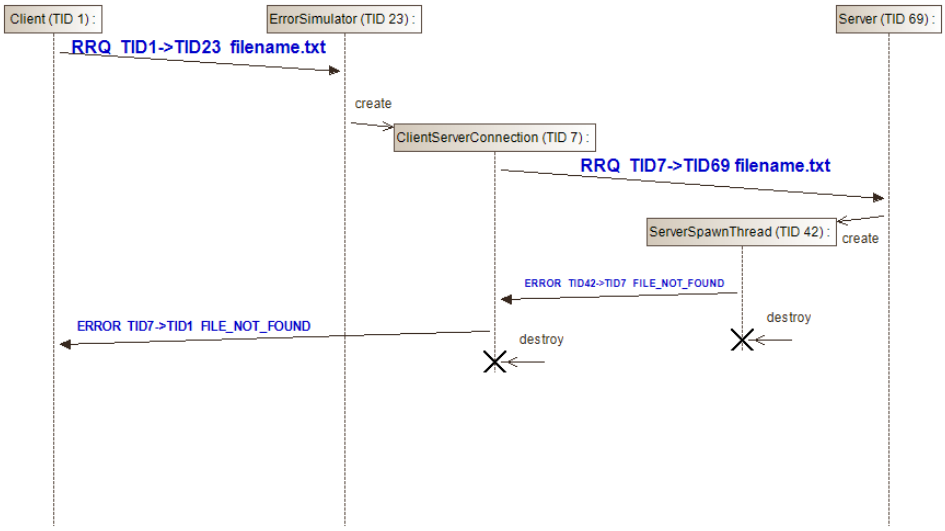



Figure 4.2.1: Attempt to read file which does not exist on server (ERROR code 1)

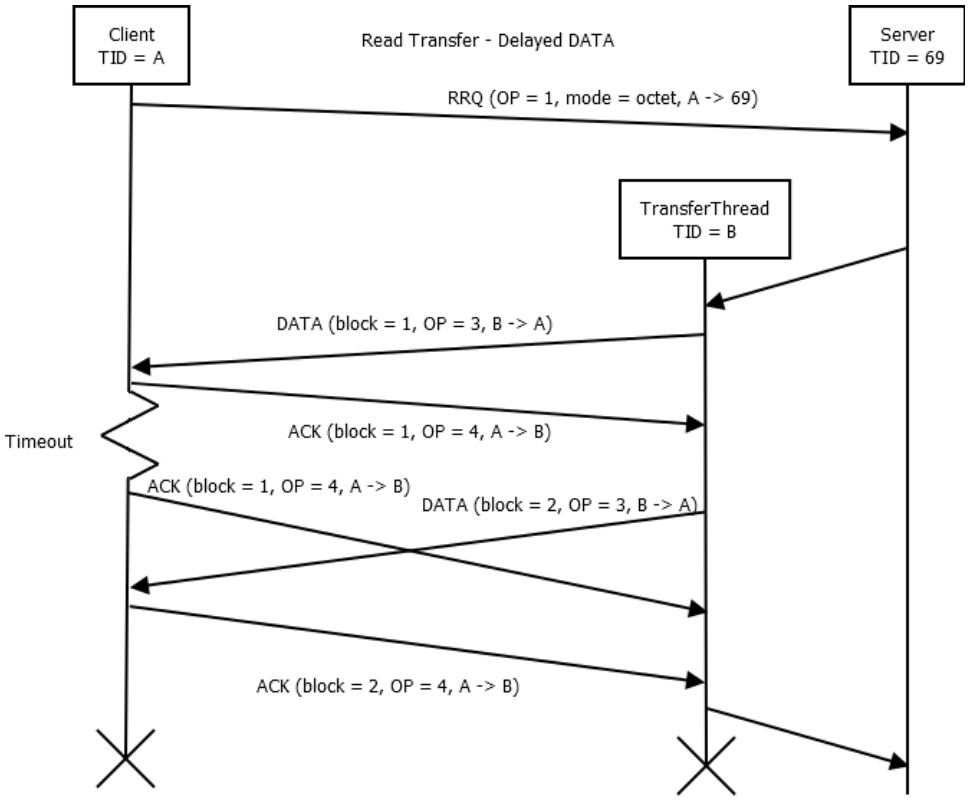



Figure 4.2.2: Read request with delayed DATA packet

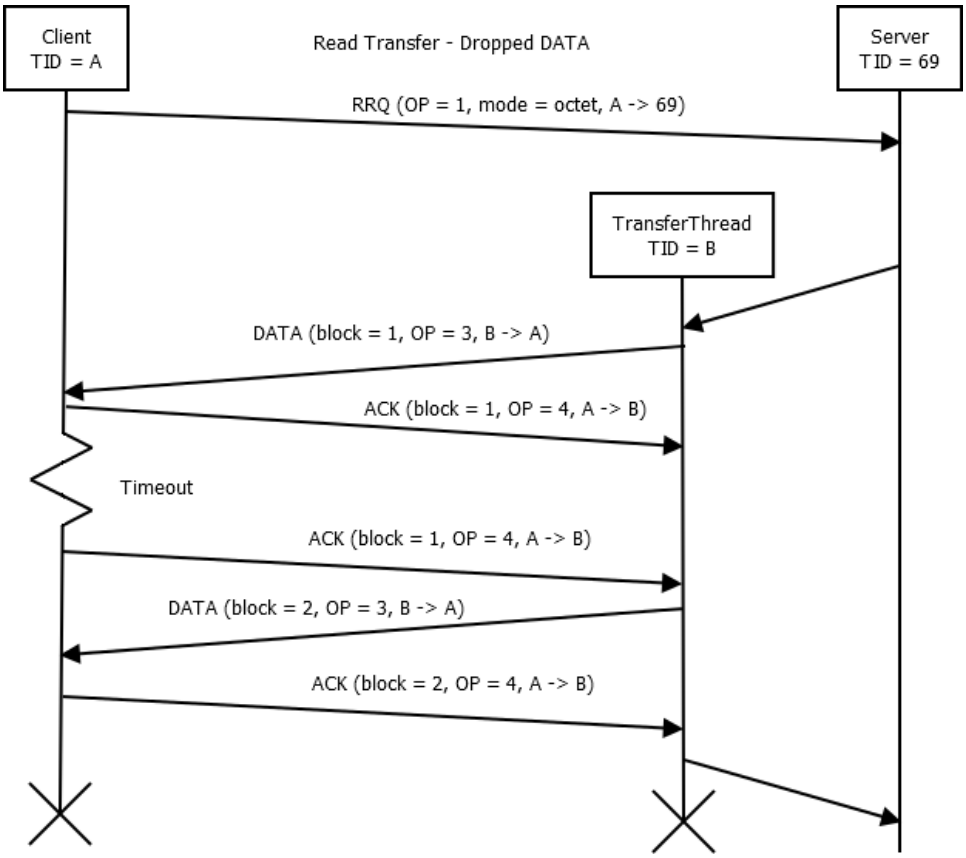



Figure 4.2.3: Read request with dropped DATA packet

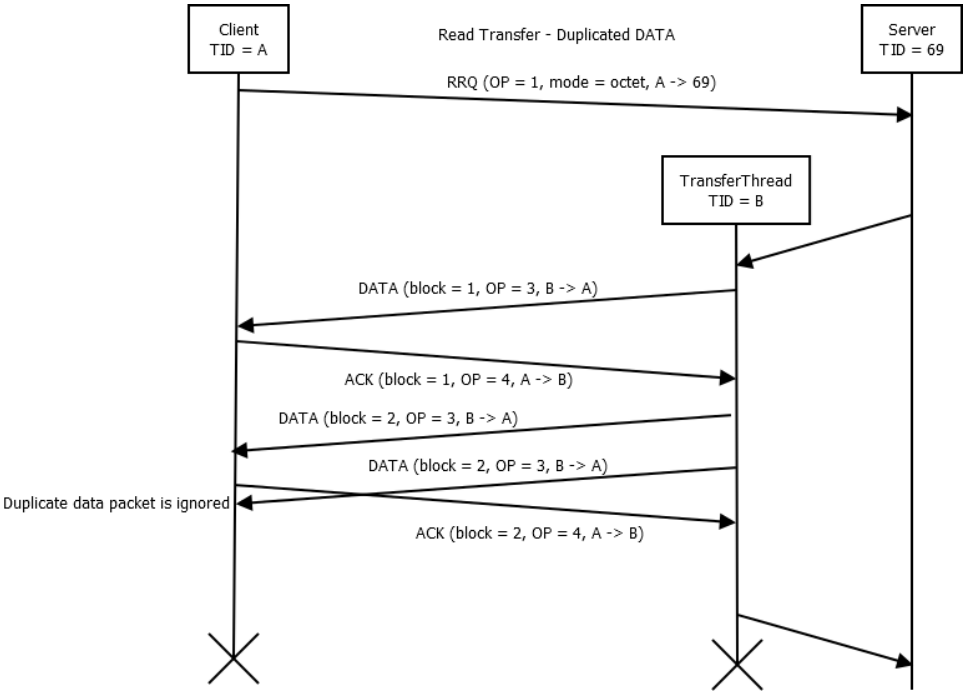


Figure 4.2.4: Read Request with duplicated DATA packet

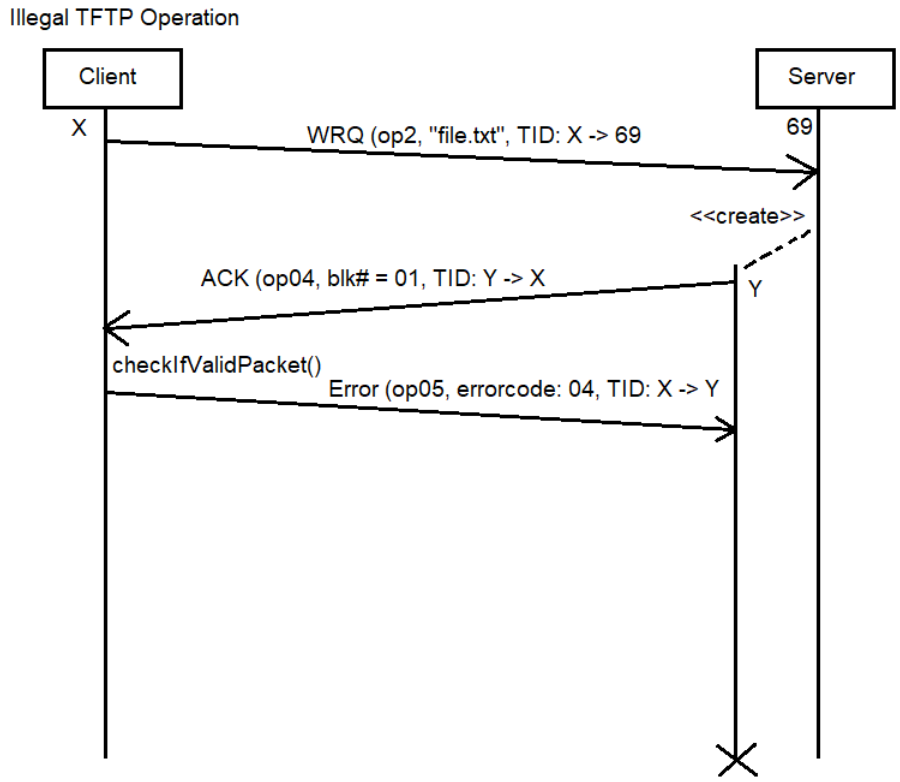



Figure 4.2.5: Write request with invalid TFTP operation (ERROR code 4)

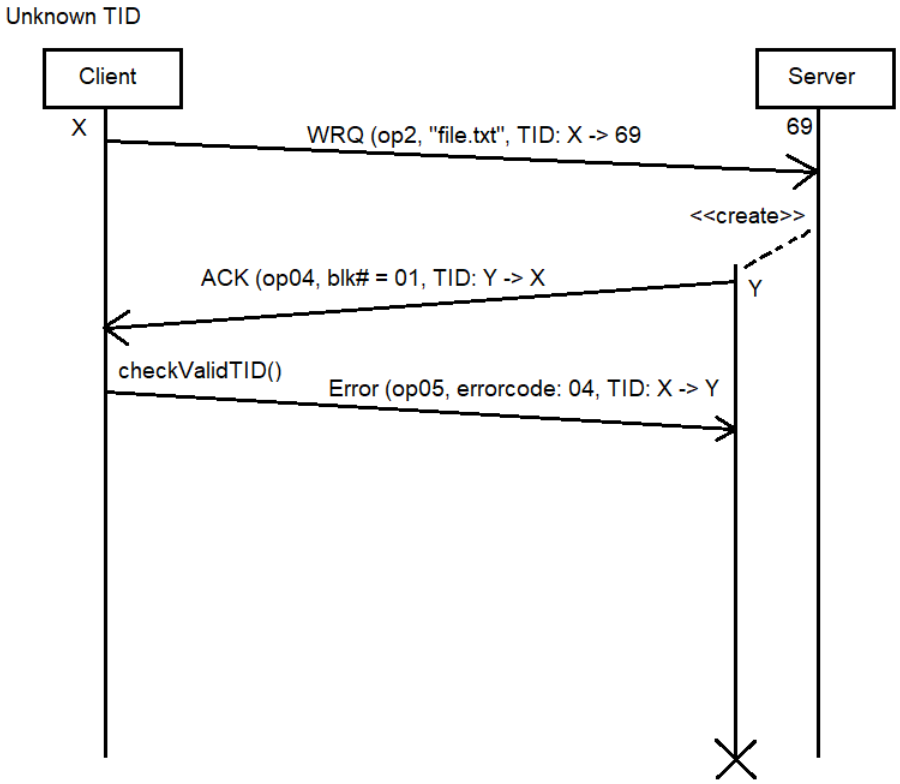



Figure 4.2.6: Write request with invalid TID (ERROR code 5)

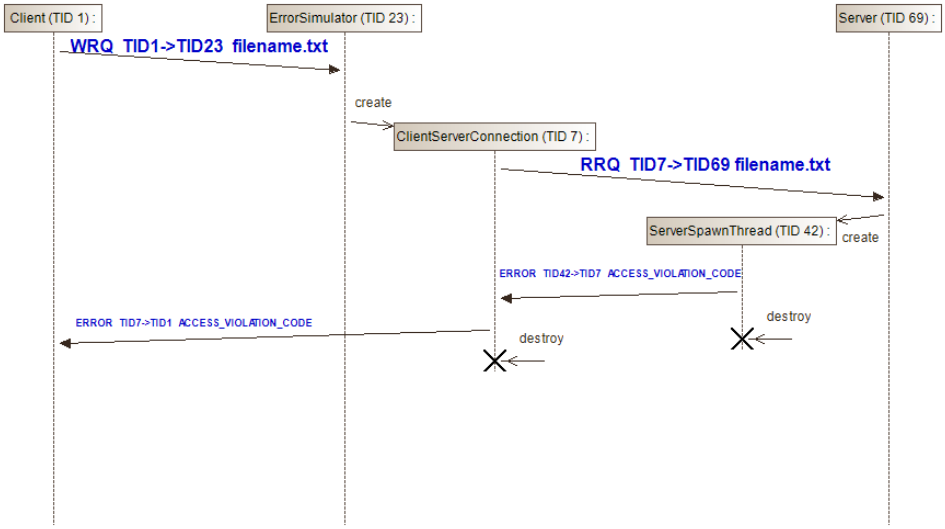



Figure 4.2.7: Write request with access violation error (ERROR code 2)

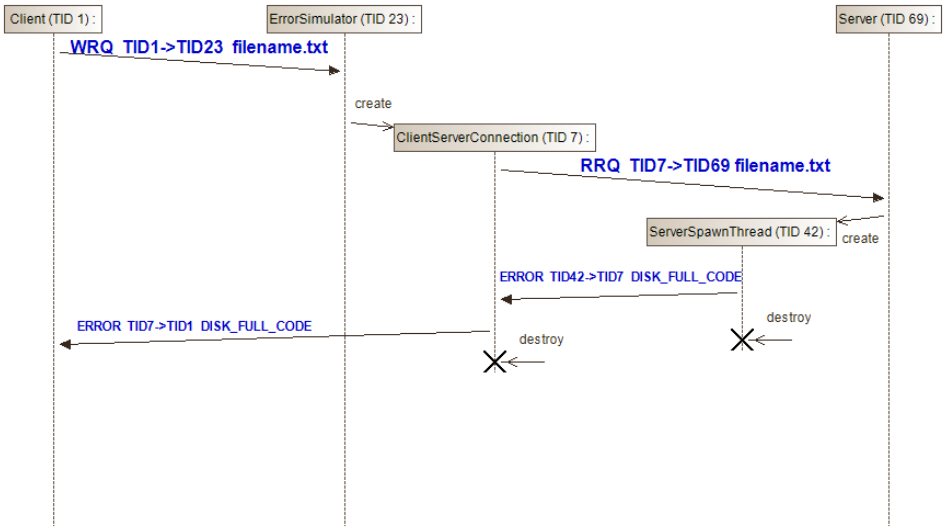



Figure 4.2.8: Write request with insufficient space to write file (ERROR code 3)

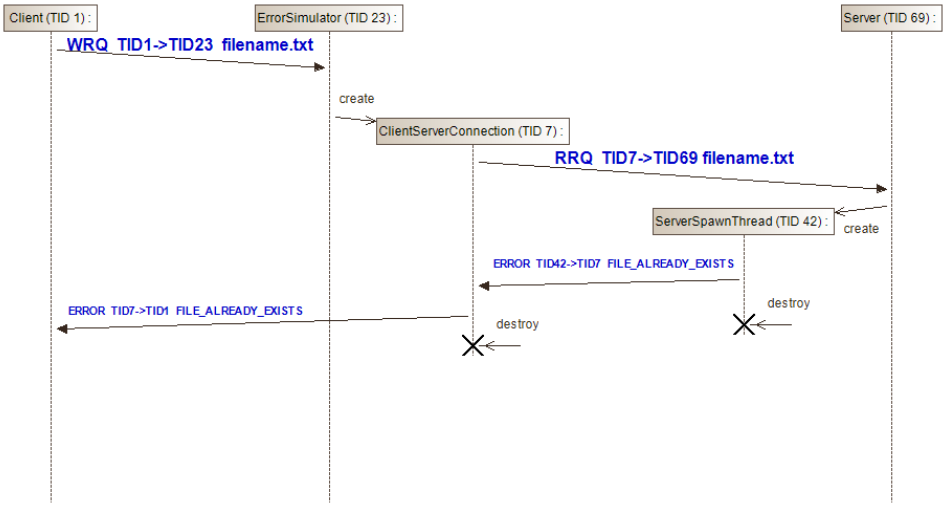



Figure 4.2.9: Write request where file being written to server already exists (ERROR code 6)

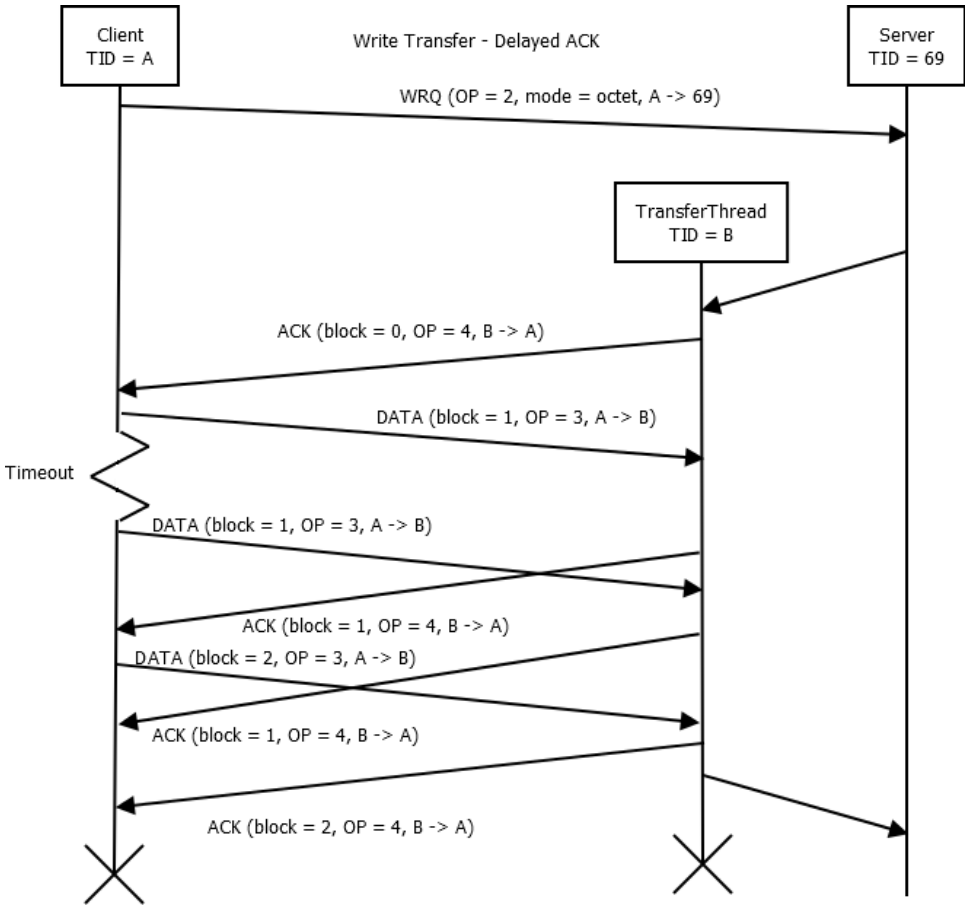



Figure 4.2.10: Write request with delayed ACK packet

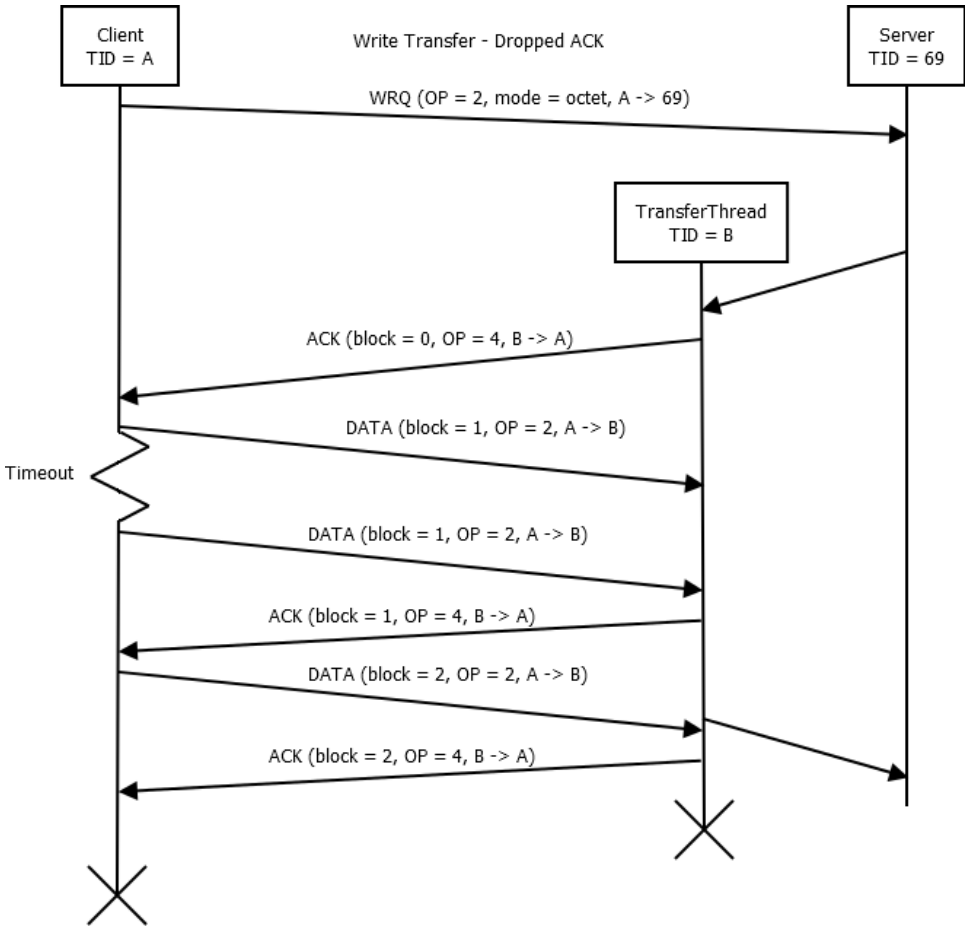



Figure 4.2.11: Write request with dropped ACK packet

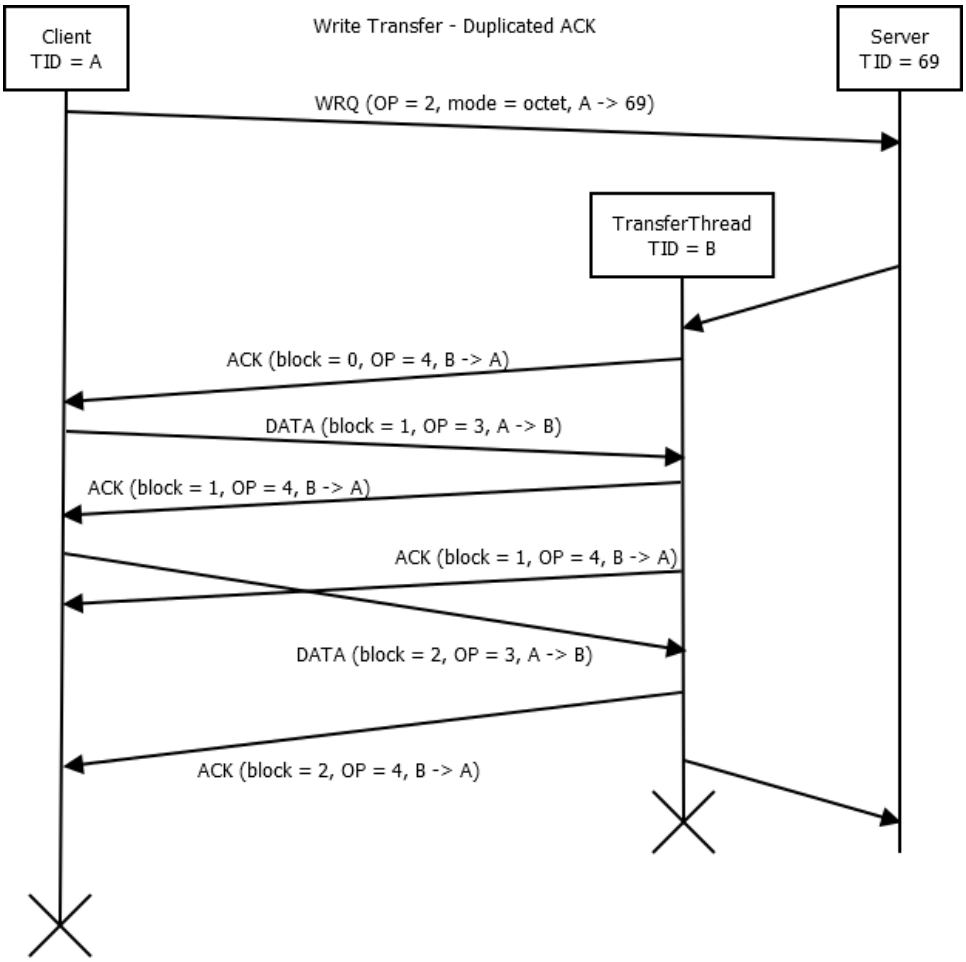



Figure 4.2.12: Write request with duplicated ACK packet

## 4.3 UCM Diagrams

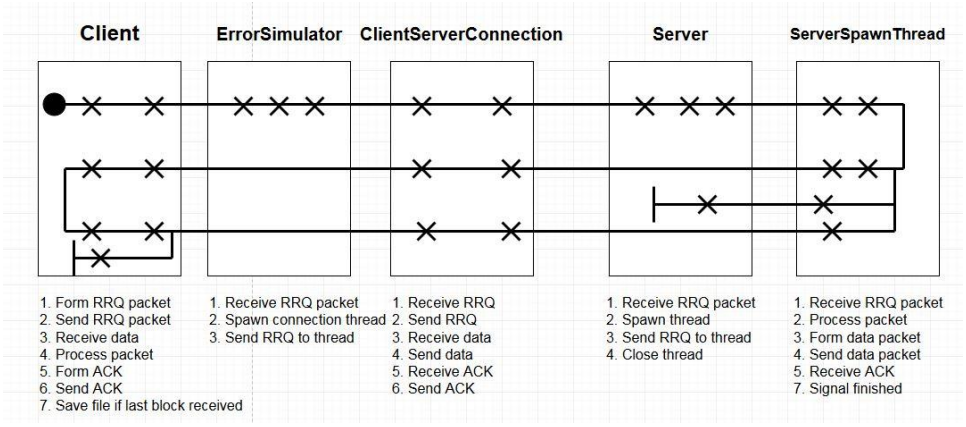



Figure 4.3.1: Read request under normal conditions

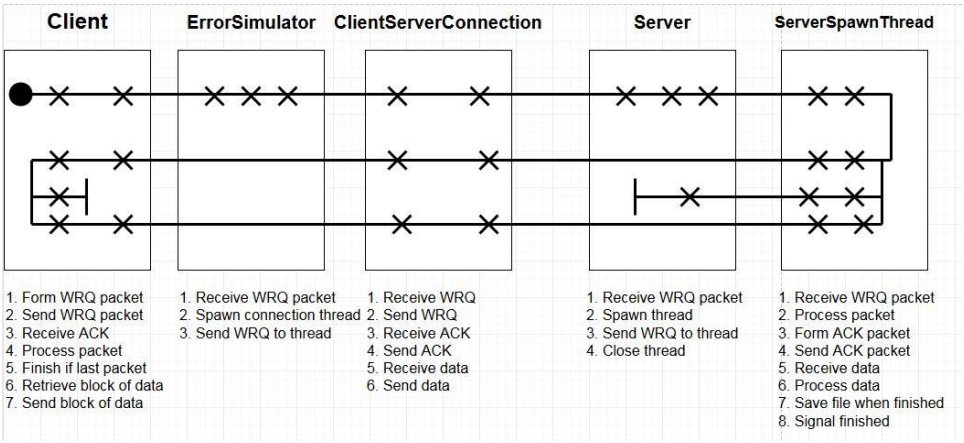



Figure 4.3.2: Write request under normal conditions