FILE TRANSFER APPLICATION
ELEN4017 Project Report

Kayla-Jade Butkow (714227) and Jared Ping (704447)

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Abstract: This paper presents the design, implementation and testing of a File Transfer Application, consisting of a server and a graphical user interface client. The developed system follows the guidelines set out in RFC 959, and exceeds the required minimum specifications. Wireshark was used to obtain results, and from the collected packets, it was clear that the request and response messages follow the correct order. The implemented server is able to interact with a standard FTP client, and the client is able to interact with a standard FTP server. The server is also able to handle multiple simultaneous control connections and data connections. Future reccomendations include the implementation of multiple transmission modes and file structures.

Key words: client, file transfer protocol, server, Wireshark

1. INTRODUCTION

The File Transfer Protocol (FTP) is essential in the implementation of a File Transfer Application. The File Transfer Protocol allows for the transfer of files between two end systems [1]. FTP runs on top of TCP and, uniquely, makes use of two TCP connections: a control connection and a data connection [1]. The FTP commands and replies are sent over the control connection, which remains open for the full duration of the client's connection to the server [1]. The data connection is non-persistent, and is created when the transfer of data is required [1]. A File Transfer Application consists of an FTP server and an FTP client. This report presents the design, implementation and testing of a File Transfer Application, including an overview of the system, details of the implemented code, results and a critical analysis of the system. The division of labour between group members is also discussed.

2. SYSTEM OVERVIEW

The implemented system follows the guidelines set out in RFC 959 - File Transfer Protocol (FTP) [2], which specifies the minimum required FTP implementation as well as the full list of FTP commands and replies.

2.1 FTP Server

The FTP server runs from the user's local host and allows users to connect through a locally hosted FTP client, as well as through FTP clients on the same network. The server is also hosted on the *Google Cloud* platform and is thus accessible from any network using the IP address *35.195.1.55*. Through its user management system, the server facilitates the storage of files within a user's unique repository. It is capable of running on any Unix based system.

The server's user management system functions by requesting users to authenticate themselves upon connecting to the server. This information is used to provide each user with their own file repository. Each user's repository is maintained to ensure that any changes that are made, are kept from one session to the next. The repository implementation provides the user exclusive access to their files and prevents users from being aware of other user repositories, thus providing a secure and user-tailored experience. Unauthenticated users are unable to perform any modification operations such as uploading or deleting files or directories.

The server has been created in accordance with the RFC 959 standards (as defined in [2]) to allow for compatibility with standard FTP clients. This allows users to connect to the server from a range of FTP clients. An extensive number of RFC commands have been implemented to provide improved compatibility and functionality for FTP clients which utilise the additional commands. A server logger has also been implemented to track client requests in real-time for server monitoring and debugging capabilities.

Multi-threading has been utilised in the development of the server to facilitate simultaneous client connections. This supports both control and data connections, thus allowing for multiple users to connect, browse directories, and upload and download files concurrently.

Unimplemented Features: The file structure type of the server is defaulted to *file*, and the *page* and *record* structures were not implemented, Furthermore, the data transmission mode of the server is defaulted to the *stream* mode, and the *block* and *compression* modes were not implemented. These features were not implemented on account of the complexity and time overhead of the features. Since the implemented structure type and transmission mode are the defaults, as specified by RFC 959, no client compatibility issues will be encountered [2].

2.2 FTP Client

The FTP client runs from the user's local host and allows the user to interact with the FTP server in order to transfer files. In order to improve user experience, a client with a graphical user interface (GUI) was implemented.

The client allows the user to specify the FTP server address that they wish to connect to, as well as the port that the server is running on. The user is also able to input their username and password for the FTP server.

Once the user has successfully connected to the FTP server, they are able to view their local file system as

well as their remote file system within the client GUI. The user is also able to navigate both file systems. Once the required file is found, the user is able to upload the file to the remote server from the local file system, or download the file from the remote system to the local storage. When uploading a file, the file is saved to the currently selected directory on the server. If a directory has not been selected by the user, the file is saved to the home directory of the user's remote repository. Likewise, when the user is downloading a file, the file is saved to the current local folder, or if none is selected, to the user's home directory. On Mac OS X operating systems, this home directory is found at */Users/Username*. If a file is selected rather than a directory, the downloaded file is saved in the directory in which the selected file is found.

The user also has the ability to delete files or folders, as well as to recursively remove a folder and all of its contents. Finally, the user is able to create a folder on the server in the base directory of their choosing. If a file is selected rather than a base directory, the new directory is saved in the directory containing the selected file. Once the client has finished using the FTP connection, they can disconnect from the server and connect to another server if they wish to.

Unimplemented Features: The feature to change the file structure from *file* to *record* or *page* was not implemented on account of the complexity of the implementation. Since the *file* structure is the default type, any server that the client wishes to interact with will be compatible with the *file* structure type [2]. The client also does not allow the user the opportunity to change the transmission mode from *stream* to *block* or *compression*. Once again, since stream mode is the default mode, any FTP server must accept *stream* mode, meaning that implementation of the other types is unnecessary [2]. The client also does not have implementation to allow the user to append data onto the end of an existing text file. Finally, the client does not cater for the renaming of files and folders.

3. COMMANDS AND REPLY CODES

There are five groups of reply codes determined by the first digit of the three digit code [2]. These groups indicate whether the response is positive or negative and allow the client to make an informed decision as to its next course of action.

The groups are as follows:

1xx	Positive Preliminary Reply
2xx	Positive Completion Reply
3xx	Positive Intermediate reply
4xx	Transient Negative Completion reply
5xx	Permanent Negative Completion reply

At least one reply code from each group has been implemented. A list of the implemented commands and the reply codes is given in *Table B1*. The table also contains a brief description of each command.

4. IMPLEMENTATION DETAILS

The server and the client were both implemented using Python 3. On both systems, all communication sockets are created using the Python `socket` module [3]. The sending and receiving of messages are performed using methods from this module. Interfacing with the operating system is performed using the `os` module [4]. This module allows for the traversing of paths in the operating system, as well as for saving and opening files [4].

Communication between the server and client is performed through the establishment of a TCP connection. This TCP connection acts as a control connection to transfer FTP commands and replies between the client and the server [1]. When sending FTP commands to the server, the messages are formatted using the format in *Figure 1*. In the figure, SP indicates a space and CRLF is the end of line sequence (`\r\n`). All communicated commands utilise `UTF-8` encoding to ensure communication compatibility.

Command	SP	Arguments	CRLF

Figure 1: FTP Command Format

4.1 Server

In order to host client connections, the server listens for any incoming connections. These are then established as a TCP connection between the two endpoints. A `serverListener()` function was created for this purpose with sockets configured to utilise the `SO_REUSEADDR` argument. This ensures that active client connections are maintained in the event of a sudden server restart.

Each new incoming client connection was handled by binding the connection to a new thread, thus allowing the server to handle simultaneous connections. The instantiated thread waits for client transmission using the `recv()` function. The received message is then decoded and the requested command is executed. The resulting response code is encoded and sent back to the client using the `send()` function. The server utilises responses to provide the client with a result corresponding to the requested command. The aforementioned process is illustrated in *Figure A1*. These responses indicate command successes, errors, mode changes, and other relevant indicators. The response codes are described in Section 3.

Configuration: The server provides a range of configuration commands for clients to utilise based on how they are attempting to interact with the server. `PASV` specifies for the server Data Transfer Process (DTP) to listen on a non-default data port, specified by the server, and to wait for a connection rather than initiate one upon receipt of a transfer command [2]. `PORT` offers similar functionality however, the port is explicitly specified by the client. `MODE` specifies the data transfer method to be utilised by the server. `STRU` specifies the type of file

structure to be used for representation of data by the server.

File browser: The server provides a wide range of commands that offer the functionality of file browser commands in a terminal client. All of the commands mentioned in this section return information or execute actions on the user's remote file system. `PWD` prints the path of the current working directory. `CWD` allows for the client to change their current working directory path to another location. `CDUP` allows for the client to change the current working directory to the parent directory. `MKD` allows the client to make a directory in the specified path. `RMD` allows the client to remove the specified directory. `DELE` allows the client to remove a specified file from the current directory. `LIST` provides a list of content contained within the current working directory. A custom utility was created for the `LIST` function to provide the client with comprehensive information about the directory contents such as file permissions, file size and user group. `RNFR` and `RNTO` allows a client to rename a folder or file on the server.

File transfer: The server provides comprehensive functionality to handle file uploads and downloads. This begins with allowing clients to specify the file type through the `TYPE` command, thus allowing the differentiation between ASCII and binary file transfer. `RETR` and `STOR` facilitate a file being downloaded and uploaded from the server respectively. Furthermore, `APPE` allows a file to be uploaded and its content appended to the file if it already exists on the server. If not, it serves the same purpose as `STOR`.

Miscellaneous: In addition to the existing server infrastructure, additional commands exist to improve the user experience. `NLST` returns only the names of content within a directory. `REST` specifies the server marker at which a file transfer is to be restarted. `SYST` returns the operating system of the server host. `HELP` returns the list of available server commands and their parameters to be supplied for successful use. `NOOP` functions as a connection testing command to ensure the client is still connected to the server. Finally, `QUIT` closes the client connection.

4.2 Client

In order to connect to the server, once the user has supplied the server address and port, a TCP connection is created between the server and the client. To communicate with the server, a `send()` function was created which takes in a string containing the FTP command, a space and the arguments. The end of line sequence is then appended to the string and the resulting string is transmitted to the server. The use of this function ensures that all messages sent to the server have the correct format. Once any control message has been sent to the server, the client receives the response, and decodes it into a string in the `receive()` function. To allow the user to see the responses from the server, all received responses are printed onto a message log on the GUI. In order to ensure that the `receive()` function is called after every message is sent, an `action()` function was created which calls the `send()` function and then the `receive()` function. This is to ensure that there is no mismatch between the command sent and the reply received.

Before uploading or downloading a file, the client sends a `PASV` command, which requests that the server creates a new data port and listens on that port for a connection from the client [2]. As a response to the command, the server sends the client the IP address and port number of the new socket. The port number, which is a 16 bit number, is sent to the client as two eight bit numbers [2]. The port is therefore calculated by multiplying the first number (the most significant byte) by 256 and adding the result to the second number [2]. Thereafter, the client connects to the port so that data can be transferred.

Uploading files: In order to upload a file, it is necessary to inform the server of whether an ASCII or binary file (image type) is being transmitted, so that the correct encoding can be used. In order to determine the type of the file to be uploaded, the `magic` module is used. The module determines the type of a file by classifying the file's headers [5]. If the type is found to be text, `TYPE A` is sent to the server. Otherwise, `TYPE I` is transmitted. Thereafter, a `STOR` command is sent to the server along with the full path of the file to be uploaded. Thereafter, the file is uploaded to the server. During the upload process, the file is divided up into chunks and each chunk transmitted to the server. A flow chart detailing the upload process is given in *Figure A2*.

Downloading files: When downloading files, it is again necessary to specify the file type. Since the files lie on the server, the `magic` module could not be used to determine the file type. Rather, the file type was deduced from the file extension, using the `mimetypes` module. This file type is then compared to a list of ASCII file types, and if the file type is found in the list, `TYPE A` is sent to the server. Otherwise, `TYPE I` is transmitted. Once the file type has been sent, a `RETR` command is sent along with the full path of the file to be downloaded. A new file, with the filename of the file to be downloaded, is then opened locally. Chunks of data are received by the client and then written to the open file. Once no more data is received, the file is closed and the download is completed.

Deleting folders and files and making folders: The user is able to delete a file or folder on the remote system. They do so by selecting the file or folder and then pressing the *Delete* button. The client then uses the method described below to determine whether the user is trying to delete a file or a folder. If a file is to be deleted, the `DELE` command is sent to the server. Likewise, for

a folder, the `RMD` command is sent. Both of these commands are followed by the full path to the item to be deleted. The user is also able to create a folder by pressing the *Create Directory* button. The pressing of this button prompts the user to input the name of the new folder. This folder is created using the `MKD` command, which is sent along with the path to the new directory.

Differentiating between files and folders: In many instances in the client, it is necessary to differentiate between a folder and a file on the server. Once such example of this is in deciding whether a `DELE` or `RMD` command should be sent, as described above. In order to differentiate, the response codes of the `CWD` command are used. If the response to a `CWD` command has a `550` code, it implies that the path points to a file and not a folder. If the response has a `250`, the path points to a folder. Thus, this method is used as a differentiator wherever one is needed.

GUI: The client was implemented as a GUI using the PyQt4 module. The GUI provided a simple user interface consisting of push buttons that allow the user to perform functions such as uploading and downloading files, and two file systems. The file systems of the server and client were created by taking the current path in the file system and creating a directory item for each of the directories in the path. The final directory is then populated with the folders and files contained in it. For the server file system, this information was obtained using the `PWD` and `LIST` commands. For the client file system, the information was obtained using the `walk` method of the `os` module. In order to change directories in the remote file system, a `CWD` command is sent along with the path to the directory of interest.

5. DIVISION OF WORK

Since the FTP server has two clear parts, the server and the client, the work was divided accordingly. Jared Ping wrote all of the code for the server, as well as the sections in the report pertaining to the server. Kayla-Jade Butkow wrote the code for the client, as the sections of the report related to the client. Kayla-Jade also wrote the section pertaining to the commands and reply codes and the introduction, while Jared detailed the structure of the code and wrote the conclusion. The results and critical analysis sections and the abstract were written by the partners together.

6. RESULTS

In order to test the system, it was necessary to test all of the implemented functionality on both the server and the client, as well as to test the interaction between the server and a standard FTP client, and between the client and a standard FTP server. This functionality was tested by performing actions on the various clients and then viewing the messages sent and responses receiving using Wireshark.

The most important functionality to be tested is the ability to log into the server, navigate through the remote file system and upload and download files. Furthermore, the ability to create a directory, delete files and directories and to log out of the server also required testing. Tests of all of these functions were performed for each of the above-mentioned interactions.

6.1 Interaction between the implemented FTP server and client

Figure C1 presents an image of the client interacting with the implemented server. From the image, it is clear that there is a unique repository for the user, and that the user is unable to navigate to other user's repositories.

Section C1 provides Wireshark excerpts of all of the implemented server client interactions. The excerpts prove that the client and server are compatible and that all of the required FTP functionality can be performed.

Figure C12 shows that the server is able to handle multiple simultaneous connections. It can also be seen from *Figure C13* that simultaneous uploads and downloads from different users can be performed.

6.2 Interaction between a standard FTP server and the implemented client

In testing this interaction, it was necessary to ensure that the created client and the standard server were compatible and that all of the functionality implemented in the client functioned correctly when paired with a standard server. The server used can be found at *ELEN4017.ug.eie.wits.ac.za*.

Wireshark screenshots indicating this functionality is given in *Section C2*. From the images in *Section C2*, it is evident that all functionality works as expected when paired with a standard FTP server.

6.3 Interaction between the implemented FTP server and a standard FTP client

The testing of this interaction is essential to ensure that the server can interact with clients other than the one created. This interaction was performed using the Mac OS X client, *ForkLift*.

The results of the tests for this interaction are given in *Section C3*. These images again prove that the server is compatible with a standard FTP client.

7. CRITICAL ANALYSIS

An analysis of the successes and limitations of the implemented system is given below.

7.1 Successes

The system is a fully functional, stable and well implemented solution. As proven in *Section 6.*, it fulfils all of the requirements for a file transfer application, namely:

- A client and a server that are able to meet all of the requirements of a minimal FTP implementation, as defined in [2], including server reply messages and error handling
- A client with a simple user interface and that is able to interact with a standard FTP server
- A server that maintains repositories for different users and that is able to interact with a standard FTP client
- A server that can handle multiple clients simultaneously performing data transfer operations using multi-threading
- The ability to upload and download various file types
- The use of Wireshark to obtain results
- The ability to use the system when the client and the server lie on different hosts within the same network

The system also performs all of these actions without the use of any high-level FTP libraries. Furthermore, both the server and the client implement features beyond those mentioned in the minimum FTP implementation, which is regarded as a large success of the system. It was stipulated that five reply code should be implemented, however on account of the large number of features, many reply code were implemented (as indicated in *Table B1*). This allows for a more informative and complete system, and is also seen as a success. Another large success of the project is aspect of multi-threading within the server. Not only is the server able to maintain control connections with multiple clients simultaneously, it is also able to handle multiple simultaneous upload and download processes. This enhances the user friendliness of the system, as multiple users can use it at the same time without their processes being affected. Finally, since the server is hosted using the *Google Cloud* Platform, it is accessible from any network which is regarded as a large success.

7.2 Limitations

The largest limitation of the implemented system is that the client only functions correctly on Mac OS X operating systems. This is a limitation as it reduces the number of users who are able to use the developed client.

A limitation of the server is that it does not have the functionality to implement a file structure other than *file*, nor a data transmission mode other than *stream*. The implications of this is that a standard FTP client will be required to use the default mode and structure, which may limit the functionality of the client. Data transmissions cannot be automatically restarted when using the *stream* transmission mode. The server's ability to handle simultaneous data transfer translates to higher network traffic as the number of connected clients increases. The server's available bandwidth is dependent on the host network and thus packet queuing can occur during peak traffic. The lack of compression data transmission mode results in a lower available bandwidth for very large network transmissions [2].

Since the append command is never sent by the client, if the user tries to upload a file with a name that already exists in the current directory, the pre-existing file will be overwritten. This could result in the accidental loss of the user's data. Another limitation lies within the file systems in the client GUI. After a file or directory has been modified, it does not update automatically. It needs to be reselected in order for the modifications to be loaded.

7.3 Future Development

For future development, the server should be enhanced to handle different file structures and transmission modes. The client should implement an automatic refresh every time a file or folder is modified. Furthermore, the functionality of the client should be enhanced to cater for more FTP commands.

8. CODE STRUCTURE AND PREREQUISITES

8.1 Code Structure

Both the server and client utilise a class based code structure. The server features a single class which utilises the thread class to instantiate server connection instances from clients. Class based methods are used to manage class properties and handle received client commands. Methods, external to the class, are used to provide logging capabilities for the server as well as implement a listener to handle incoming client connections. A utility class has been created to manage file properties and is included for use by the server. The client features a single class which contains all relevant methods to handle communication with the server.

8.2 Application Prerequisites

The File Transfer Application requires *Python 3* and external python modules are included by the client to provide extensive functionality. The `Magic` module and the `mimetypes` module are utilised to determine the file type of selected files. The `PyQt4` module is used to provide a GUI implementation for the client, greatly enhancing the user experience. The `macports` package manager can be used to assist with setting up the required modules. This is done by running the following commands in the terminal:

```
port install file port install py3-pyqt4
```

The server can then be initialised by running the following command:

```
sudo python ftpserver.py
```

The use of `sudo` is due to the server running on command port 21 which requires administrative privileges. Finally, the client can be initialised by running the following command:

```
python GUIClient.py
```

9. CONCLUSION

The design, implementation and testing of a File Transfer Application was presented. The system was deemed to be a success since it met all of the basic requirements of a File Transfer Application, and also implemented many additional features. Through the use of Wireshark, it is clear that the system implemented all of the required FTP reply codes and that the codes and responses are sent in the correct order. For future development, more FTP commands should be implemented in order to develop a complete File Transfer Application.

REFERENCES

[1] J. Kurose and K. Ross. *Computer Networking. A Top-Down Approach*, p. 51. Pearson Education, sixth ed., 2013.

[2] J. Postel and J. Reynolds. "File Transfer Protocol (FTP)." *Network Working Group*, oct 1985.

[3] Python. "socket - Low-level networking interface." URL `https://docs.python.org/3.6/library/socket.html`. Last accessed: 17/03/2018.

[4] Python. "os - Miscellaneous operating system interfaces." URL `https://docs.python.org/3.6/library/os.html`. Last accessed: 17/03/2018.

[5] A. Hupp. "python-magic.", 2001. URL `https://github.com/ahupp/python-magic`. Last accessed: 17/03/2018.

Appendix

A Algorithms

A1 Server Algorithm

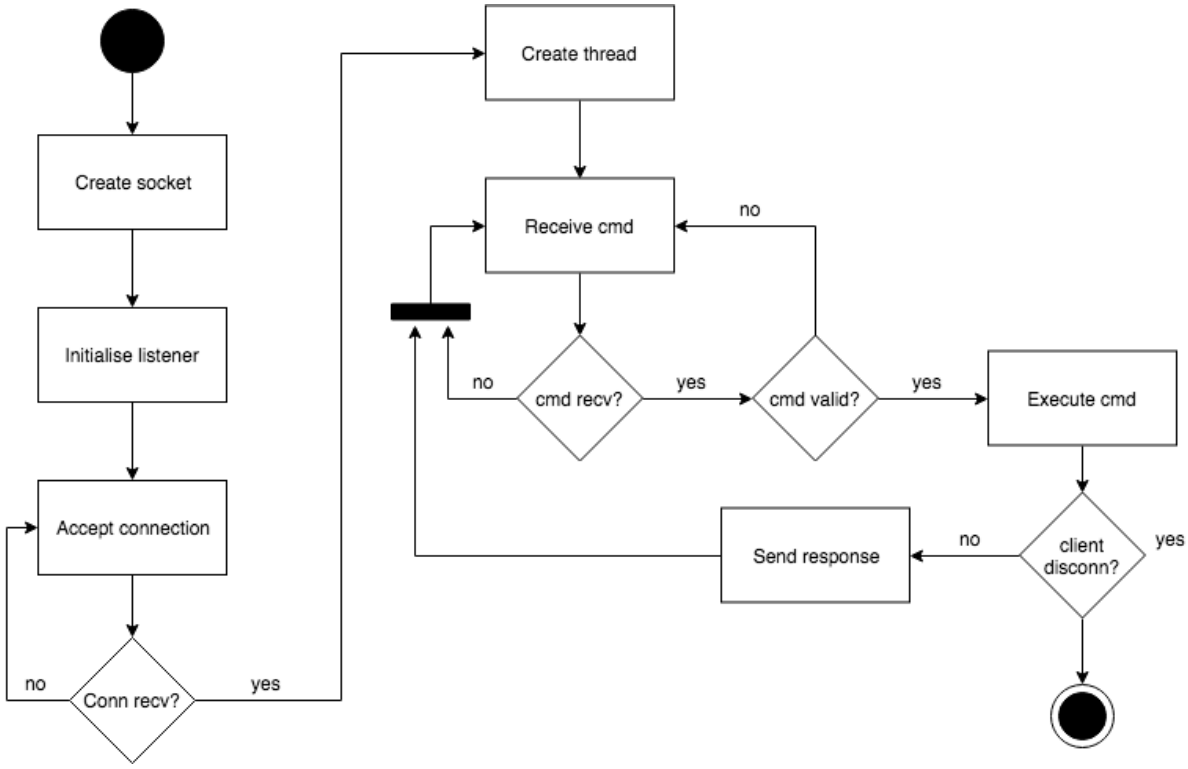


Figure A1: Flow chart depicting the process of server handling connections and client requests

A2 Client Algorithm

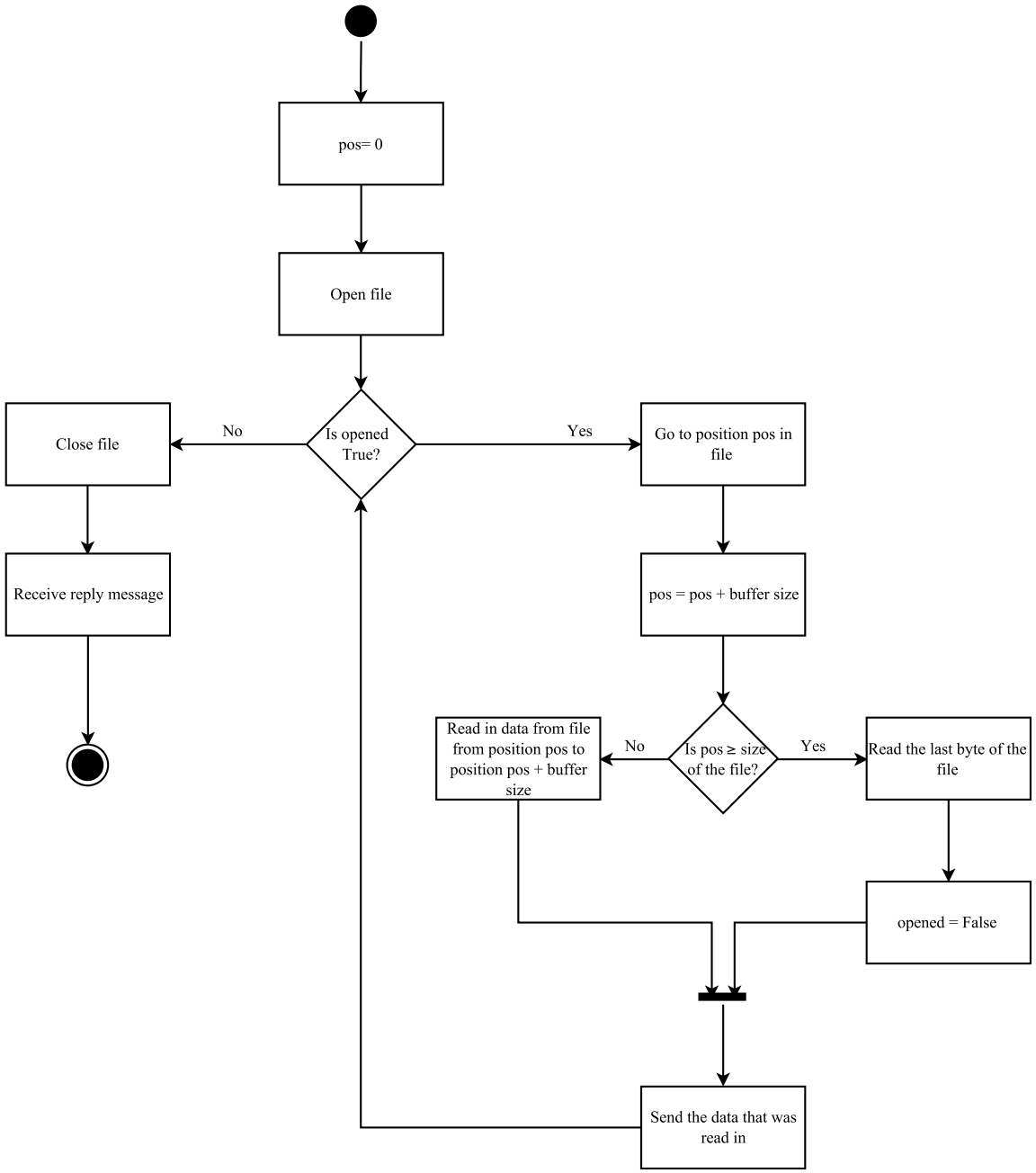


Figure A2: Flow chart depicting the process of uploading a file to the server

B Implemented Commands and Reply Codes

Table B1: Table detailing the implemented commands and reply codes

Command	Description	Reply Code
USER	Allows the user to input their username in order to be authenticated	501 Syntax error in parameters or arguments. 331 User name okay, need password.
PASS	Allows the user to input their password for authentication	501 Syntax error in parameters or arguments. 503 Bad sequence of commands. 230 User logged in, proceed.
TYPE	Its argument is used to specify the file type of the file to be retrieved or stored	200 Binary file mode. 200 Ascii file mode. 501 Syntax error in parameters or arguments.
PASV	Requests that the server listens on a new data port and waits for a connection	227 Entering Passive Mode (*IP Address, Port*)

MODE	Its argument is used to specify the data transfer type. Only stream transfer mode was implemented	200 Stream transfer mode. 502 Command not implemented. 501 Syntax error in parameters or arguments.
STRU	Its argument is used to specify the file structure of the file to be retrieved or stored. Only the File type was implemented	200 File Structure = File. 502 Command not implemented.
STAT	The command causes a status response to be sent over the control connection. The functionality for this command was not implemented	502 Command not implemented.
PORT	The argument specifies the data port to be used in the data connection	200 Get port.
LIST	Returns a list of the contents of a directory including permissions. The argument is used to specify the path in which the contents should be returned. If an argument is not provided, a list of the contents of the current working directory is supplied	530 User not logged in. 550 LIST failed Path name not exists. 150 Here is listing. 226 List done.
NLST	Returns a list of the names of the contents of a directory	530 User not logged in. 550 NLST failed Path name not exists. 150 Here is listing. 226 List done.
CWD	Changes the working directory of the server. The argument is used to specify the new working directory.	550 CWD failed. Directory does not exist. 250 CWD Command successful.
PWD	Returns the current working directory	257 *Path to current working directory*
CDUP	Changes the working directory to the parent of the current directory.	200 OK.
DELE	Deletes a file off the remote host. Its argument is used to specify the file to be deleted	530 User not logged in. 550 DELE failed File *file name* does not exist. 450 DELE failed delete not allowed. 250 File deleted.
MKD	Makes a file on the remote host. Its argument specifies the path to the new file and the name of the file to be created	530 User not logged in. 257 Directory created. 550 MKD failed. Directory "*directory name*" already exists.
RMD	Deletes a directory off the remote host. Its argument is used to specify the directory to be deleted	530 User not logged in. 450 Invalid permissions. 250 Directory deleted.
RNFR	Its argument specifies a file to be renamed	550 RNFR failed. File or Directory *file or directory name* does not exist. 350 RNFR successful - awaiting RNTO
RNTO	Its argument specifies the new name of file. The file to be renamed was indicated using the RNFR command prior to calling the RNTO command	550 RNTO failed. File or Directory *file or directory name* does not exist. 250 RNTO successful
REST	The argument field represents the checkpoint at which the file transfer is to be restarted.	250 File position reset.
RETR	This command causes the server to send a copy of a file over the data connection. The argument specifies the name of the file to be downloaded	150 Opening data connection. 226 Transfer complete.

STOR	This command causes the server to save a copy of a file that is sent over the data connection. The argument specifies the name of the file that is being uploaded	530 STOR failed. User is not logged in. 150 Opening data connection. 226 Transfer completed.
APPE	This command causes the server to save a copy of a file that is sent over the data connection. If the file name exists at the path on the server, data is appended to the file. Otherwise, a new file is created	530 APPE failed. User is not logged in. 150 Opening data connection. 226 Transfer completed.
SYST	Used to find the server's operating system type	215 *server operating system* type.
NOOP	Prompts a 200 OK response from the server.	200 OK.
HELP	Displays help information	
QUIT	Terminates the control connection between the user and the server	221 Goodbye.

C Results

This section contains all of the results collected during the testing of the system. This includes Wireshark excerpts and an image of the client GUI.

C1 Interaction between the implemented FTP server and client

This section contains images and Wireshark screenshots (*Figure C1* to *Figure C11*) depicting the interaction between the implemented client and server when running on two different hosts within the same network.

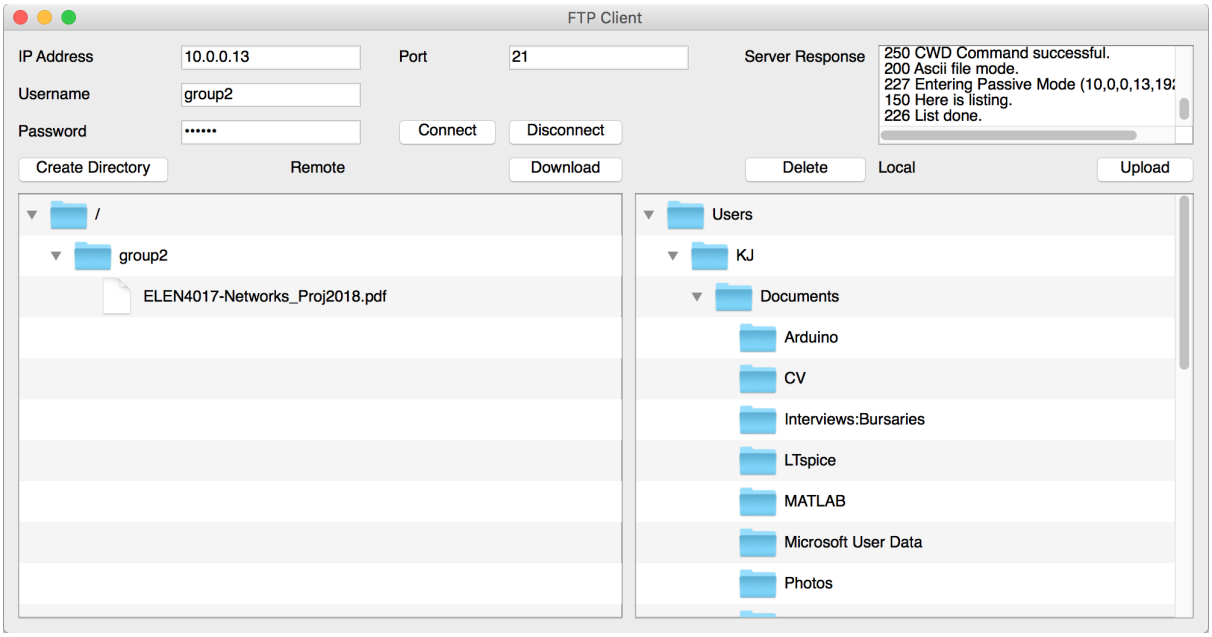


Figure C1: GUI client interacting with the implemented FTP server

```
    17 13:24:04.861285    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP      99 Response: 220 Service ready for new user.
    19 13:24:04.862199    Kayla-Jades-MacBook-Pro.local 192.168.66.185                FTP      79 Request: USER group2
    21 13:24:04.863846    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP     102 Response: 331 User name okay, need passwor
    23 13:24:04.864707    Kayla-Jades-MacBook-Pro.local 192.168.66.185                FTP      79 Request: PASS group2
    25 13:24:04.866528    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP      96 Response: 230 User logged in, proceed.
```

Figure C2: FTP commands and replies when logging in

```
    31 13:24:04.873438    Kayla-Jades-MacBook-Pro.local 192.168.66.185                FTP      71 Request: PWD
    33 13:24:04.876044    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP      83 Response: 257 "/group2/".
    35 13:24:04.876302    Kayla-Jades-MacBook-Pro.local 192.168.66.185                FTP      74 Request: TYPE A
    37 13:24:04.906390    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP      88 Response: 200 Ascii file mode.
    39 13:24:04.907795    Kayla-Jades-MacBook-Pro.local 192.168.66.185                FTP      72 Request: PASV
    41 13:24:04.913040    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP     119 Response: 227 Entering Passive Mode (192,16
    46 13:24:04.926678    Kayla-Jades-MacBook-Pro.local 192.168.66.185                FTP      72 Request: LIST
    49 13:24:04.929609    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP      88 Response: 150 Here is listing.
    54 13:24:04.932084    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP      82 Response: 226 List done.
    62 13:24:08.335793    Kayla-Jades-MacBook-Pro.local 192.168.66.185                FTP      79 Request: CWD /group2
    65 13:24:08.374035    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP      95 Response: 250 CWD Command successful.
    67 13:24:08.426233    Kayla-Jades-MacBook-Pro.local 192.168.66.185                FTP      71 Request: PWD
    69 13:24:08.531104    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP      82 Response: 257 "/group2".
```

Figure C3: FTP commands and replies when navigating the remote file system

```
  1409 13:25:36.655789    Kayla-Jades-MacBook-Pro.local 192.168.66.185                FTP      74 Request: TYPE A
  1411 13:25:36.657232    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP      88 Response: 200 Ascii file mode.
  1413 13:25:36.658249    Kayla-Jades-MacBook-Pro.local 192.168.66.185                FTP      72 Request: PASV
  1415 13:25:36.659799    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP     119 Response: 227 Entering Passive Mode (192,168,66,185,230,
  1420 13:25:36.662529    Kayla-Jades-MacBook-Pro.local 192.168.66.185                FTP      71 Request: PWD
  1423 13:25:36.665167    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP      87 Response: 257 "/group2/Test".
  1425 13:25:36.665401    Kayla-Jades-MacBook-Pro.local 192.168.66.185                FTP     115 Request: STOR /group2/Test/gaussianEliminationOutput.txt
  1427 13:25:36.670412    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP      96 Response: 150 Opening data connection.
  1435 13:25:36.719733    192.168.66.185                Kayla-Jades-MacBook-Pro.local FTP      91 Response: 226 Transfer completed.
```

Figure C4: FTP commands and replies when uploading an ASCII type file

C2 Interaction between a standard FTP server and the implemented client

Wireshark screenshots (*Figure C14* to *Figure C23*) depicting the interaction between the created client and the standard FTP server are given in this section.

293	13:24:35.662141	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	74	Request: TYPE I
295	13:24:35.834411	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	89	Response: 200 Binary file mode.
297	13:24:35.835424	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	72	Request: PASV
299	13:24:35.852870	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	119	Response: 227 Entering Passive Mode (192,168,66,185,230,
304	13:24:35.879624	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	71	Request: PWD
307	13:24:35.885720	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	87	Response: 257 "/group2/Test".
309	13:24:35.885905	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	116	Request: STOR /group2/Test/ELEN4017-Networks_Proj2018.pd
311	13:24:35.889471	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	96	Response: 150 Opening data connection.
1049	13:24:36.637524	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	91	Response: 226 Transfer completed.

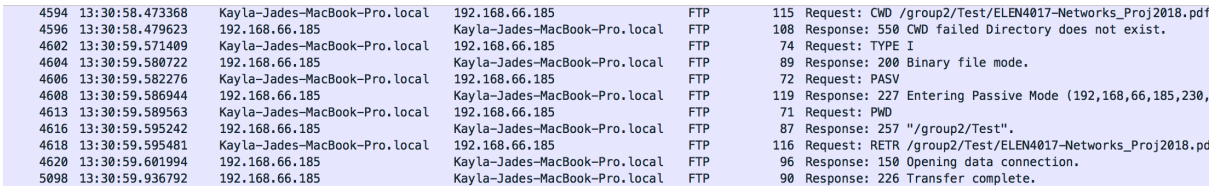
Figure C5: FTP commands and replies when uploading an image type file

5299	13:31:12.993482	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	114	Request: CWD /group2/Test/gaussianEliminationOutput.txt
5302	13:31:12.999283	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	108	Response: 550 CWD failed Directory does not exist.
5304	13:31:14.053304	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	74	Request: TYPE A
5306	13:31:14.072136	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	88	Response: 200 Ascii file mode.
5308	13:31:14.073190	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	72	Request: PASV
5310	13:31:14.076615	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	119	Response: 227 Entering Passive Mode (192,168,66,185,230,
5315	13:31:14.081264	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	71	Request: PWD
5319	13:31:14.085012	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	87	Response: 257 "/group2/Test".
5321	13:31:14.085184	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	115	Request: RETR /group2/Test/gaussianEliminationOutput.txt
5323	13:31:14.088680	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	96	Response: 150 Opening data connection.
5329	13:31:14.091999	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	90	Response: 226 Transfer complete.

Figure C6: FTP commands and replies when downloading an ASCII type file

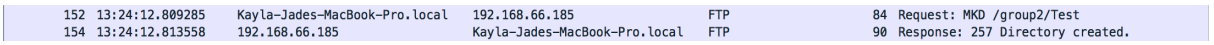
C3 Interaction between the implemented FTP server and a standard FTP client

This section contains Wireshark excerpts depicting the interaction between the server and a standard client when running on two different hosts within the same network. The images for this section are found in *Figure C24* to *Figure C30*.



4594	13:30:58.473368	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	115	Request: CWD /group2/Test/ELEN4017-Networks_Proj2018.pdf
4596	13:30:58.479623	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	108	Response: 550 CWD failed Directory does not exist.
4602	13:30:59.571409	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	74	Request: TYPE I
4604	13:30:59.580722	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	89	Response: 200 Binary file mode.
4606	13:30:59.582276	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	72	Request: PASV
4608	13:30:59.586944	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	119	Response: 227 Entering Passive Mode (192,168,66,185,230,
4613	13:30:59.589563	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	71	Request: PWD
4616	13:30:59.595242	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	87	Response: 257 "/group2/Test".
4618	13:30:59.595481	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	116	Request: RETR /group2/Test/ELEN4017-Networks_Proj2018.pd
4620	13:30:59.601994	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	96	Response: 150 Opening data connection.
5098	13:30:59.936792	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	90	Response: 226 Transfer complete.

Figure C7: FTP commands and replies when downloading an image type file

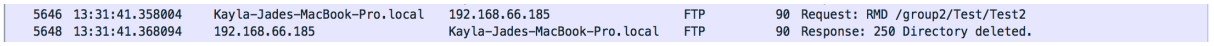


152	13:24:12.809285	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	84	Request: MKD /group2/Test
154	13:24:12.813558	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	90	Response: 257 Directory created.

Figure C8: FTP commands and replies when creating a directory

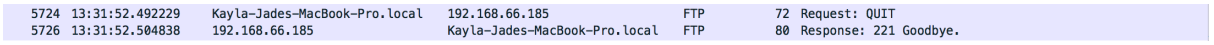
5454	13:31:29.059067	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	115	Request: CWD /group2/Test/ELEN4017-Networks_Proj2018.pdf
5456	13:31:29.064493	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	108	Response: 550 CWD failed Directory does not exist.
5458	13:31:29.115752	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	116	Request: DELE /group2/Test/ELEN4017-Networks_Proj2018.pd
5460	13:31:29.126617	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	85	Response: 250 File deleted.

Figure C9: FTP commands and replies when deleting a file



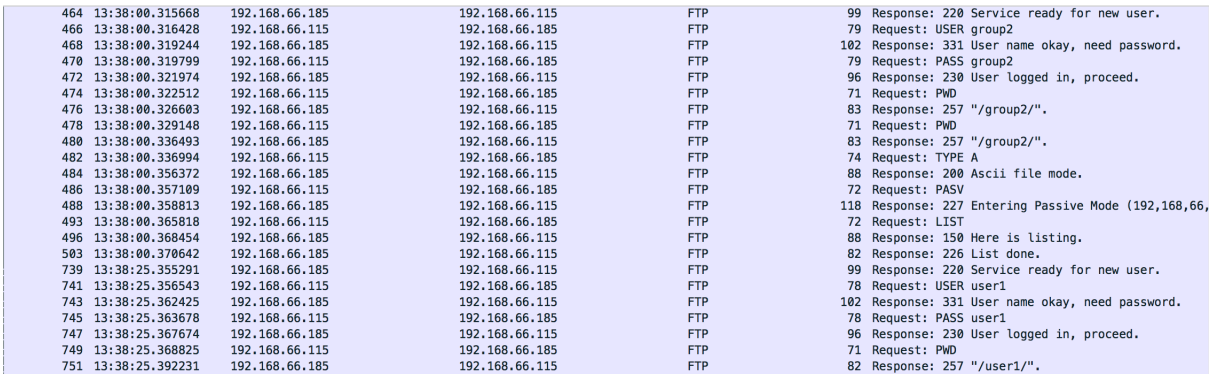
5646	13:31:41.358004	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	90	Request: RMD /group2/Test/Test2
5648	13:31:41.368094	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	90	Response: 250 Directory deleted.

Figure C10: FTP commands and replies when deleting a directory



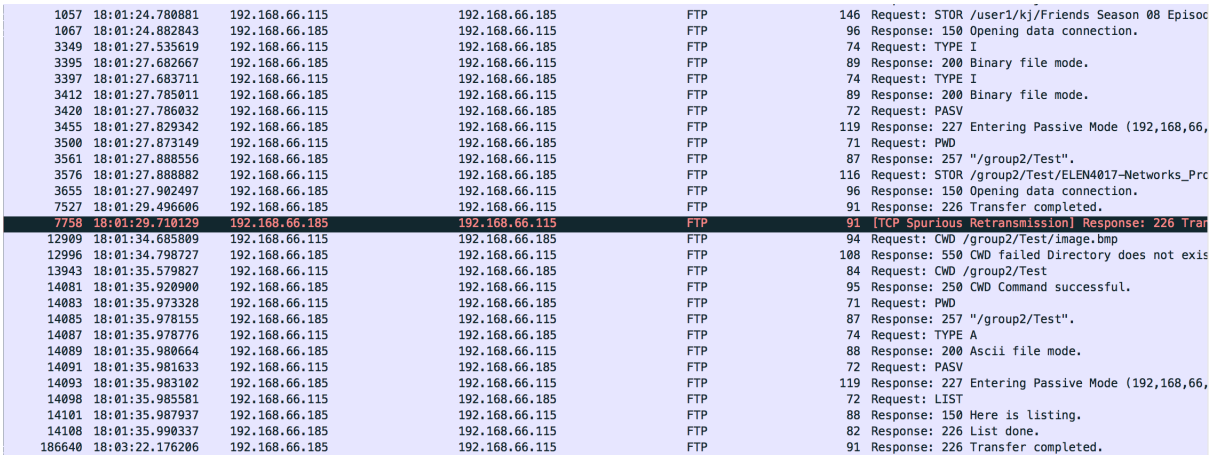
5724	13:31:52.492229	Kayla-Jades-MacBook-Pro.local	192.168.66.185	FTP	72	Request: QUIT
5726	13:31:52.504838	192.168.66.185	Kayla-Jades-MacBook-Pro.local	FTP	80	Response: 221 Goodbye.

Figure C11: FTP commands and replies when logging out from the server



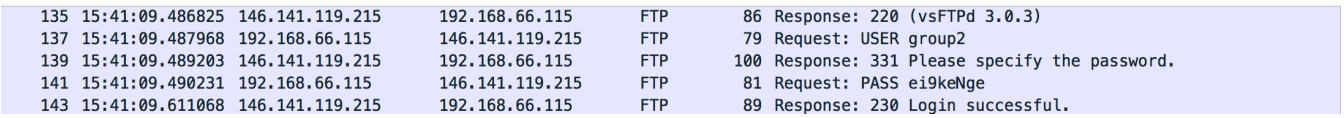
464	13:38:00.315668	192.168.66.185	192.168.66.115	FTP	99	Response: 220 Service ready for new user.
466	13:38:00.316428	192.168.66.115	192.168.66.185	FTP	79	Request: USER group2
468	13:38:00.319244	192.168.66.185	192.168.66.115	FTP	102	Response: 331 User name okay, need password.
470	13:38:00.319799	192.168.66.115	192.168.66.185	FTP	79	Request: PASS group2
472	13:38:00.321974	192.168.66.185	192.168.66.115	FTP	96	Response: 230 User logged in, proceed.
474	13:38:00.322512	192.168.66.115	192.168.66.185	FTP	71	Request: PWD
476	13:38:00.326603	192.168.66.185	192.168.66.115	FTP	83	Response: 257 "/group2/".
478	13:38:00.329148	192.168.66.115	192.168.66.185	FTP	71	Request: PWD
480	13:38:00.336493	192.168.66.185	192.168.66.115	FTP	83	Response: 257 "/group2/".
482	13:38:00.336994	192.168.66.115	192.168.66.185	FTP	74	Request: TYPE A
484	13:38:00.356372	192.168.66.185	192.168.66.115	FTP	88	Response: 200 Ascii file mode.
486	13:38:00.357109	192.168.66.115	192.168.66.185	FTP	72	Request: PASV
488	13:38:00.358813	192.168.66.185	192.168.66.115	FTP	118	Response: 227 Entering Passive Mode (192,168,66,
493	13:38:00.365818	192.168.66.115	192.168.66.185	FTP	72	Request: LIST
496	13:38:00.368454	192.168.66.185	192.168.66.115	FTP	88	Response: 150 Here is listing.
503	13:38:00.370642	192.168.66.185	192.168.66.115	FTP	82	Response: 226 List done.
739	13:38:25.355291	192.168.66.185	192.168.66.115	FTP	99	Response: 220 Service ready for new user.
741	13:38:25.356543	192.168.66.115	192.168.66.185	FTP	78	Request: USER user1
743	13:38:25.362425	192.168.66.185	192.168.66.115	FTP	102	Response: 331 User name okay, need password.
745	13:38:25.363678	192.168.66.115	192.168.66.185	FTP	78	Request: PASS user1
747	13:38:25.367674	192.168.66.185	192.168.66.115	FTP	96	Response: 230 User logged in, proceed.
749	13:38:25.368825	192.168.66.115	192.168.66.185	FTP	71	Request: PWD
751	13:38:25.392231	192.168.66.185	192.168.66.115	FTP	82	Response: 257 "/user1/".

Figure C12: Multiple clients connected simultaneously to the FTP server



1057	18:01:24.780881	192.168.66.115	192.168.66.185	FTP	146	Request: STOR /user1/kj/Friends Season 08 Episoc
1067	18:01:24.882843	192.168.66.185	192.168.66.115	FTP	96	Response: 150 Opening data connection.
3349	18:01:27.535619	192.168.66.115	192.168.66.185	FTP	74	Request: TYPE I
3395	18:01:27.682667	192.168.66.185	192.168.66.115	FTP	89	Response: 200 Binary file mode.
3397	18:01:27.683711	192.168.66.115	192.168.66.185	FTP	74	Request: TYPE I
3412	18:01:27.785011	192.168.66.185	192.168.66.115	FTP	89	Response: 200 Binary file mode.
3420	18:01:27.786032	192.168.66.115	192.168.66.185	FTP	72	Request: PASV
3455	18:01:27.829342	192.168.66.185	192.168.66.115	FTP	119	Response: 227 Entering Passive Mode (192,168,66,
3500	18:01:27.873149	192.168.66.115	192.168.66.185	FTP	71	Request: PWD
3561	18:01:27.888556	192.168.66.185	192.168.66.115	FTP	87	Response: 257 "/group2/Test".
3576	18:01:27.888882	192.168.66.115	192.168.66.185	FTP	116	Request: STOR /group2/Test/ELEN4017-Networks_Prc
3655	18:01:27.902497	192.168.66.185	192.168.66.115	FTP	96	Response: 150 Opening data connection.
7527	18:01:29.496606	192.168.66.185	192.168.66.115	FTP	91	Response: 226 Transfer completed.
7758	18:01:29.710129	192.168.66.185	192.168.66.115	FTP	91	[TCP Spurious Retransmission] Response: 226 Trar
12909	18:01:34.685809	192.168.66.115	192.168.66.185	FTP	94	Request: CWD /group2/Test/image.bmp
12996	18:01:34.798727	192.168.66.185	192.168.66.115	FTP	108	Response: 550 CWD failed Directory does not exis
13943	18:01:35.579827	192.168.66.115	192.168.66.185	FTP	84	Request: CWD /group2/Test
14081	18:01:35.920900	192.168.66.185	192.168.66.115	FTP	95	Response: 250 CWD Command successful.
14083	18:01:35.973328	192.168.66.115	192.168.66.185	FTP	71	Request: PWD
14085	18:01:35.978155	192.168.66.185	192.168.66.115	FTP	87	Response: 257 "/group2/Test".
14087	18:01:35.978776	192.168.66.115	192.168.66.185	FTP	74	Request: TYPE A
14089	18:01:35.980664	192.168.66.185	192.168.66.115	FTP	88	Response: 200 Ascii file mode.
14091	18:01:35.981633	192.168.66.115	192.168.66.185	FTP	72	Request: PASV
14093	18:01:35.983102	192.168.66.185	192.168.66.115	FTP	119	Response: 227 Entering Passive Mode (192,168,66,
14098	18:01:35.985581	192.168.66.115	192.168.66.185	FTP	72	Request: LIST
14101	18:01:35.987937	192.168.66.185	192.168.66.115	FTP	88	Response: 150 Here is listing.
14108	18:01:35.990337	192.168.66.185	192.168.66.115	FTP	82	Response: 226 List done.
186640	18:03:22.176206	192.168.66.185	192.168.66.115	FTP	91	Response: 226 Transfer completed.

Figure C13: Multiple clients simultaneously uploading files to the FTP server



135	15:41:09.486825	146.141.119.215	192.168.66.115	FTP	86	Response: 220 (vsFTPd 3.0.3)
137	15:41:09.487968	192.168.66.115	146.141.119.215	FTP	79	Request: USER group2
139	15:41:09.489203	146.141.119.215	192.168.66.115	FTP	100	Response: 331 Please specify the password.
141	15:41:09.490231	192.168.66.115	146.141.119.215	FTP	81	Request: PASS ei9keNge
143	15:41:09.611068	146.141.119.215	192.168.66.115	FTP	89	Response: 230 Login successful.

Figure C14: FTP commands and replies when logging in

149	15:41:09.616097	192.168.66.115	146.141.119.215	FTP	71	Request: PWD
150	15:41:09.618075	146.141.119.215	192.168.66.115	FTP	100	Response: 257 "/" is the current directory
152	15:41:09.618351	192.168.66.115	146.141.119.215	FTP	74	Request: TYPE A
153	15:41:09.621553	146.141.119.215	192.168.66.115	FTP	96	Response: 200 Switching to ASCII mode.
155	15:41:09.622506	192.168.66.115	146.141.119.215	FTP	72	Request: PASV
156	15:41:09.624119	146.141.119.215	192.168.66.115	FTP	119	Response: 227 Entering Passive Mode (146,141,119,215,193,98
161	15:41:09.626864	192.168.66.115	146.141.119.215	FTP	72	Request: LIST
162	15:41:09.629701	146.141.119.215	192.168.66.115	FTP	105	Response: 150 Here comes the directory listing.
169	15:41:09.633317	146.141.119.215	192.168.66.115	FTP	90	Response: 226 Directory send OK.
193	15:41:14.380620	192.168.66.115	elen4017.ug.eie.wi…	FTP	78	Request: CWD /files
194	15:41:14.382077	elen4017.ug.eie.wi…	192.168.66.115	FTP	103	Response: 250 Directory successfully changed.
196	15:41:14.433748	192.168.66.115	elen4017.ug.eie.wi…	FTP	71	Request: PWD
197	15:41:14.435388	elen4017.ug.eie.wi…	192.168.66.115	FTP	105	Response: 257 "/files" is the current directory

Figure C15: FTP commands and replies when navigating the remote file system

1024	15:42:36.066964	192.168.66.115	elen4017.ug.eie.wi…	FTP	74	Request: TYPE A
1025	15:42:36.068311	elen4017.ug.eie.wi…	192.168.66.115	FTP	96	Response: 200 Switching to ASCII mode.
1027	15:42:36.069383	192.168.66.115	elen4017.ug.eie.wi…	FTP	72	Request: PASV
1028	15:42:36.070818	elen4017.ug.eie.wi…	192.168.66.115	FTP	120	Response: 227 Entering Passive Mode (146,141,119,215,181,11
1033	15:42:36.073271	192.168.66.115	elen4017.ug.eie.wi…	FTP	86	Request: STOR textfiles.txt
1034	15:42:36.075036	elen4017.ug.eie.wi…	192.168.66.115	FTP	88	Response: 150 Ok to send data.
1041	15:42:36.130252	elen4017.ug.eie.wi…	192.168.66.115	FTP	90	Response: 226 Transfer complete.

Figure C16: FTP commands and replies when uploading an ASCII type file

265	15:41:36.365523	192.168.66.115	elen4017.ug.eie.wi…	FTP	74	Request: TYPE I
266	15:41:36.366900	elen4017.ug.eie.wi…	192.168.66.115	FTP	97	Response: 200 Switching to Binary mode.
268	15:41:36.367947	192.168.66.115	elen4017.ug.eie.wi…	FTP	72	Request: PASV
269	15:41:36.369445	elen4017.ug.eie.wi…	192.168.66.115	FTP	120	Response: 227 Entering Passive Mode (146,141,119,215,168,13
274	15:41:36.372084	192.168.66.115	elen4017.ug.eie.wi…	FTP	107	Request: STOR 714227 - ELEN4017 Laboratory 1.pdf
275	15:41:36.374309	elen4017.ug.eie.wi…	192.168.66.115	FTP	88	Response: 150 Ok to send data.
750	15:41:36.484973	elen4017.ug.eie.wi…	192.168.66.115	FTP	90	Response: 226 Transfer complete.

Figure C17: FTP commands and replies when uploading an image type file

194…	15:43:35.624143	192.168.66.115	elen4017.ug.eie.wi…	FTP	92	Request: CWD /files/textfiles.txt
194…	15:43:35.627860	elen4017.ug.eie.wi…	192.168.66.115	FTP	99	Response: 550 Failed to change directory.
194…	15:43:36.706359	192.168.66.115	elen4017.ug.eie.wi…	FTP	74	Request: TYPE A
194…	15:43:36.710722	elen4017.ug.eie.wi…	192.168.66.115	FTP	96	Response: 200 Switching to ASCII mode.
194…	15:43:36.712182	192.168.66.115	elen4017.ug.eie.wi…	FTP	72	Request: PASV
194…	15:43:36.718027	elen4017.ug.eie.wi…	192.168.66.115	FTP	120	Response: 227 Entering Passive Mode (146,141,119,215,183,24
194…	15:43:36.723170	192.168.66.115	elen4017.ug.eie.wi…	FTP	86	Request: RETR textfiles.txt
194…	15:43:36.726238	elen4017.ug.eie.wi…	192.168.66.115	FTP	138	Response: 150 Opening BINARY mode data connection for textf
194…	15:43:36.748169	elen4017.ug.eie.wi…	192.168.66.115	FTP	90	Response: 226 Transfer complete.

Figure C18: FTP commands and replies when downloading an ASCII type file

193806	15:43:28.940…	192.168.66.1…	elen4017.ug.…	FTP	113	Request: CWD /files/714227 - ELEN4017 Laboratory 1.pdf
193807	15:43:28.964…	elen4017.ug.…	192.168.66.1…	FTP	99	Response: 550 Failed to change directory.
193809	15:43:30.654…	192.168.66.1…	elen4017.ug.…	FTP	74	Request: TYPE A
193810	15:43:30.659…	elen4017.ug.…	192.168.66.1…	FTP	96	Response: 200 Switching to ASCII mode.
193812	15:43:30.661…	192.168.66.1…	elen4017.ug.…	FTP	72	Request: PASV
193814	15:43:30.675…	elen4017.ug.…	192.168.66.1…	FTP	120	Response: 227 Entering Passive Mode (146,141,119,215,187,190).
193819	15:43:30.683…	192.168.66.1…	elen4017.ug.…	FTP	107	Request: RETR 714227 - ELEN4017 Laboratory 1.pdf
193820	15:43:30.696…	elen4017.ug.…	192.168.66.1…	FTP	162	Response: 150 Opening BINARY mode data connection for 714227 - ELEN4017 Lab
194243	15:43:31.180…	elen4017.ug.…	192.168.66.1…	FTP	90	Response: 226 Transfer complete.

Figure C19: FTP commands and replies when downloading an image type file

194…	15:43:43.070695	192.168.66.115	elen4017.ug.eie.wi…	FTP	76	Request: MKD Test
194…	15:43:43.074969	elen4017.ug.eie.wi…	192.168.66.115	FTP	93	Response: 257 "/files/Test" created
194…	15:43:44.685884	192.168.66.115	elen4017.ug.eie.wi…	FTP	78	Request: CWD /files
194…	15:43:44.689825	elen4017.ug.eie.wi…	192.168.66.115	FTP	103	Response: 250 Directory successfully changed.
194…	15:43:44.741558	192.168.66.115	elen4017.ug.eie.wi…	FTP	71	Request: PWD
194…	15:43:44.742806	elen4017.ug.eie.wi…	192.168.66.115	FTP	105	Response: 257 "/files" is the current directory

Figure C20: FTP commands and replies when creating a directory

194386	15:43:52.779…	192.168.66.1…	elen4017.ug.…	FTP	83	Request: RMD /files/Test
194387	15:43:52.780…	elen4017.ug.…	192.168.66.1…	FTP	110	Response: 250 Remove directory operation successful.

Figure C21: FTP commands and replies when deleting a directory

194439	15:43:58.311…	192.168.66.1…	elen4017.ug.…	FTP	92	Request: CWD /files/textfiles.txt
194440	15:43:58.315…	elen4017.ug.…	192.168.66.1…	FTP	99	Response: 550 Failed to change directory.
194442	15:43:58.367…	192.168.66.1…	elen4017.ug.…	FTP	93	Request: DELE /files/textfiles.txt
194443	15:43:58.369…	elen4017.ug.…	192.168.66.1…	FTP	100	Response: 250 Delete operation successful.

Figure C22: FTP commands and replies when deleting a file

194…	15:44:10.263529	192.168.66.115	elen4017.ug.eie.wi…	FTP	72	Request: QUIT
194…	15:44:10.264920	elen4017.ug.eie.wi…	192.168.66.115	FTP	80	Response: 221 Goodbye.

Figure C23: FTP commands and replies when logging out from the server

254	13:55:47.340420	192.168.66.115	192.168.66.185	FTP	99	Response: 220 Service ready for new user.
256	13:55:47.373029	192.168.66.185	192.168.66.115	FTP	77	Request: USER jeff
258	13:55:47.373364	192.168.66.115	192.168.66.185	FTP	102	Response: 331 User name okay, need password.
260	13:55:47.385931	192.168.66.185	192.168.66.115	FTP	76	Request: PASS asf
262	13:55:47.386169	192.168.66.115	192.168.66.185	FTP	96	Response: 230 User logged in, proceed.

Figure C24: FTP commands and replies when logging in

273	13:55:47.436303	192.168.66.185	192.168.66.115	FTP	71	Request: PWD
275	13:55:47.436742	192.168.66.115	192.168.66.185	FTP	81	Response: 257 "/jeff/".
277	13:55:47.439262	192.168.66.185	192.168.66.115	FTP	74	Request: TYPE A
279	13:55:47.439576	192.168.66.115	192.168.66.185	FTP	88	Response: 200 Ascii file mode.
281	13:55:47.444632	192.168.66.185	192.168.66.115	FTP	72	Request: PASV
283	13:55:47.444993	192.168.66.115	192.168.66.185	FTP	119	Response: 227 Entering Passive Mode (192,168,66,115,208,162).
289	13:55:47.477215	192.168.66.185	192.168.66.115	FTP	75	Request: LIST -a
291	13:55:47.477624	192.168.66.115	192.168.66.185	FTP	88	Response: 150 Here is listing.
296	13:55:47.516298	192.168.66.115	192.168.66.185	FTP	82	Response: 226 List done.
304	13:55:47.608222	192.168.66.185	192.168.66.115	FTP	77	Request: CWD /jeff
306	13:55:47.608726	192.168.66.115	192.168.66.185	FTP	95	Response: 250 CWD Command successful.

Figure C25: FTP commands and replies when navigating the remote file system

546	13:56:12.732916	192.168.66.185	192.168.66.115	FTP	74	Request: TYPE I
548	13:56:12.733283	192.168.66.115	192.168.66.185	FTP	89	Response: 200 Binary file mode.
551	13:56:12.750615	192.168.66.185	192.168.66.115	FTP	72	Request: PASV
553	13:56:12.750972	192.168.66.115	192.168.66.185	FTP	119	Response: 227 Entering Passive Mode (192,168,66,115,208,176).
559	13:56:12.764675	192.168.66.185	192.168.66.115	FTP	95	Request: STOR /jeff/stuff/thread.cpp
561	13:56:12.765459	192.168.66.115	192.168.66.185	FTP	96	Response: 150 Opening data connection.
568	13:56:12.782956	192.168.66.115	192.168.66.185	FTP	91	Response: 226 Transfer completed.

Figure C26: FTP commands and replies when uploading an image type file

990	13:57:07.462454	192.168.66.185	192.168.66.115	FTP	74	Request: TYPE I
992	13:57:07.462911	192.168.66.115	192.168.66.185	FTP	89	Response: 200 Binary file mode.
994	13:57:07.472918	192.168.66.185	192.168.66.115	FTP	72	Request: PASV
996	13:57:07.473284	192.168.66.115	192.168.66.185	FTP	119	Response: 227 Entering Passive Mode (192,168,66,115,208,198).
1001	13:57:07.490085	192.168.66.185	192.168.66.115	FTP	95	Request: RETR /jeff/stuff/thread.cpp
1004	13:57:07.490603	192.168.66.115	192.168.66.185	FTP	96	Response: 150 Opening data connection.
1010	13:57:07.499666	192.168.66.115	192.168.66.185	FTP	90	Response: 226 Transfer complete.
1076	13:57:27.896281	192.168.66.185	192.168.66.115	FTP	95	Request: DELE /jeff/stuff/thread.cpp
1078	13:57:27.896900	192.168.66.115	192.168.66.185	FTP	85	Response: 250 File deleted.

Figure C27: FTP commands and replies when downloading an image type file and deleting a file

1114	13:57:33.771117	192.168.66.185	192.168.66.115	FTP	87	Request: MKD /jeff/delete me
1116	13:57:33.771641	192.168.66.115	192.168.66.185	FTP	90	Response: 257 Directory created.

Figure C28: FTP commands and replies when creating a directory

1268	13:57:44.804004	192.168.66.185	192.168.66.115	FTP	87	Request: RMD /jeff/delete me
1270	13:57:44.804449	192.168.66.115	192.168.66.185	FTP	90	Response: 250 Directory deleted.

Figure C29: FTP commands and replies when deleting a directory

268	13:55:47.431432	192.168.66.185	192.168.66.115	FTP	72	Request: FEAT
270	13:55:47.431794	192.168.66.115	192.168.66.185	FTP	162	Response: 500 Syntax error, command unrecognized. This may include errors sucl

Figure C30: FTP commands and replies when an unimplemented command is called