File Transfer Protocol Application

Morongwa Thobejane (1432370)
School of Electrical and Information Engineering, University of the Witwatersrand, Johannesburg 2050, South Africa

Abstract—**The paper presents the design, implementation and testing of a file transfer protocol (FTP) applications which allows files sharing via Wifi with IPv4. Both the client and server applications are implemented and they communicate through the FTP implemented using the Python3 TCP socket library. The client and server are able to perform desired activities as per the RFC 959 document. Multi-threading was used in the download and upload commands so that multiple threads can be created for large files. The application results are inspected through Wireshark.**

July 07, 2021

I. Introduction

A computer network is a group of devices that are connected to each other in order to communicate and share data electronically. The communication between two devices can be divided into seven layers of the Open System Interconnection (OSI) model [1]. The 7^{th} layer is the application layer and it operated at the user end by interacting with user applications, such as email and file transfer. Each layer of communication has its own functions and protocols, and the application layer has the following protocols: FTP, SMPP, SNMP, HTTP and DHCP. This project focuses on the on the FTP application.

The aim of the project is to create a file transfer application system which makes use of protocols, application and transport layer and socket programming. The system has a user interface which allows a user to carry out desired tasks as per the RFC 959 document. The serve implementation is able to communicate withe client and it stores the information and repositories for the users. The client implementation does the same. The application is implemented in Python.

II. Background

The File Transfer Protocol (FTP) is one of the earliest file sharing protocols. It was first documented in the RFC 171 document and released in June 1971, which was years before HTTP was introduced [2]. The RFC 959 documents presents the changes and improvements have been made starting from the RFC 171 document. The primary objective of the FTP application is to allow a reliable and data transfer and data sharing between the client and local storage (or the remote storage) installed on the server machine. The FTP application consists of two parallel connections which are the control and data connections [3]. The control connection handles the user authentication and the commands to switch between remote directories, to upload and download files. The data connection is used to transfer files.

TABLE I
THE IMPLEMENTED COMMANDS, THEIR CODES AND REPLY MESSAGES

Code	Reply message	Command
331	Please specify password	USER
230	Login successful	PASS
200	Listing completed ASCII mode enabled	TYPE NOOP
150	File status okay; about to open data connection	LIST RETR STOR
226	Closing data connection. Requested transfer action successful	LIST RETR STOR
250	Requested file okay, completed	DELE
221	Goodbye	QUIT
227	Entering Passive Mode (10,196,37,16,50,107)	PSV
254	Cannot open data connection	PORT
501	Syntax error in parameters or arguments	TYPE

III. Implementation

Both the FTP Server and FTP Client were implemented in Python 3 because Python it provides well documented libraries which reduce complexity. The application focus on implementing the basic commands listed in the RFC 959 document in order to make both the server and client sides work.Threading is used in order to create multiple threads to download and upload large files. Wireshark was used to inspect the FTP replies. Table ones shows the implemented commands and reply message codes, as per sections 4.2 and 5.1 of the RFC 959 document.

A. Server

The server uses multi-threading to handle multiple clients that connect to it at the same time. A TCP socket is initiated with a dynamically assigned IP address on port 21 and then the server waits and listens for new clients to connect. If a client connects to the server successfully, a thread is created to handle the client's requests, which are the FTP commands as per the RFC 959 document. The server only handles commands if it is in active mode, which means that a control connection is established from client to server and a data connection is established from server to client. The client requests are handled in 5 stages and they occur in the following order:

- Read requests from socket using a buffer size of 8192 bytes.
- Split the requests into FTP commands and its corresponding parameters, if there are any.
- Identify the FTP command and call a relevant function corresponding to the implementation of the command.

- Send a response back to the client.

When a client connects to the server, user authentication is required in order for the client to be able to access file in the remote file system, because the server does not support anonymous client mode. The server assigns each user a home directly which is identified by the client user name and it can only be accessed by the specific user.

The commands of the FTP system are implemented in a threading class, as a list of standalone functions. Each of these functions correspond to one of the commands shown in table... and they follow a general procedure which involves the validation of parameters, the performing of the requested task and returning a response message which is encoded and sent to the client by a handler thread. The **RETR** and **STOR** functions create threads which allow the transfer of large files, multiple downloads or uploads of files. The implementation of the threads allows prevent the client from downloading a file which is currently being uploaded and it also prevents the client from uploading a file that is currently being downloaded. The required 5 reply codes groups were implemented and the data type that is being used is the binary type.

IV. Client

The FTP client implementation has 3 sections, which are the Graphical user Interface (GUI), the socket layer and the logic layer. The GUI layer was created using the QT designer tool whereby the exists a window which consists of components. The QT designer tool was used because it helps to avoid having to use code to define the size, position and orientation of each component, which would make the source complex and hard to read. The GUI is used as a separate file which is the linked to the Python programming file (back-end). The GUI is made up of 9 features which are shown in table...

The Python socket library was used to establish a TCP control and data connections. The socket connects to server address and port which are both obtained from the user. FTP works in two modes which are the active and passive mode and this is defined by how the connection of the socket is established. In active mode, the client address which is usually the IP address of the computer which is running the client side and it also sends a random port number. In passive mode the socket object of the client connects only to the IP address and port of the server. The client socket object then connects to the server on the port.

In the client logic section, an object orientated programming approach was implemented whereby the client is in the form of a class called FTP_client. The approached was used because is allows for the client commands to be grouped as activities which are in one function and this makes the client implementation easier. It only takes a single button click to fetch data from the GUI when a user wants to login and the username, password, IP address and port of the server are required. Once the data has been fetched, the commands: USER, PASS and LIST are sent to the server with the required parameters and the the login procedure is completed. Without object orientated programming, the commands would have to be implemented in a form of functions.

V. Results

The client can successfully connect and log on to the server with the correct information (username, password, server IP address and port number) provided. The client can is able to download files from the server and store the files in folder named ”Downloaded_files” and this folder is in the same location as the client python file. The client is also able to upload file into the folder that the server is running from. The client can upload and download different formats of files such as video, images and music files. The client can list files and folders in the server, move up and down folder levels and create ans delete files on the server. The client can also switch between passive and active connections. The server allows multiple clients to connect with the help of multi-threading and it was able to function as intended with FTP GUI client called FileZilla. It can perform all the commands requested by the client successfully. The server can connect to the coded FTP client and successfully communicate and share data even when they are running on separate computers. Figure 1, in Appendix A shows the results observed on Wireshark.

VI. Results Analysis

The client and server both meet the RFC 959 minimum implementation standards as per section 5.1. The implementation does not have a user interface which was supposed to be implemented on QT designer and this is because there were many problems encountered in the process of designing and implementing the user interface. These problems could be resolved due to limited time. The application still functions properly and it performs all its commands without the UI as shown in figure 1.

VII. conclusion

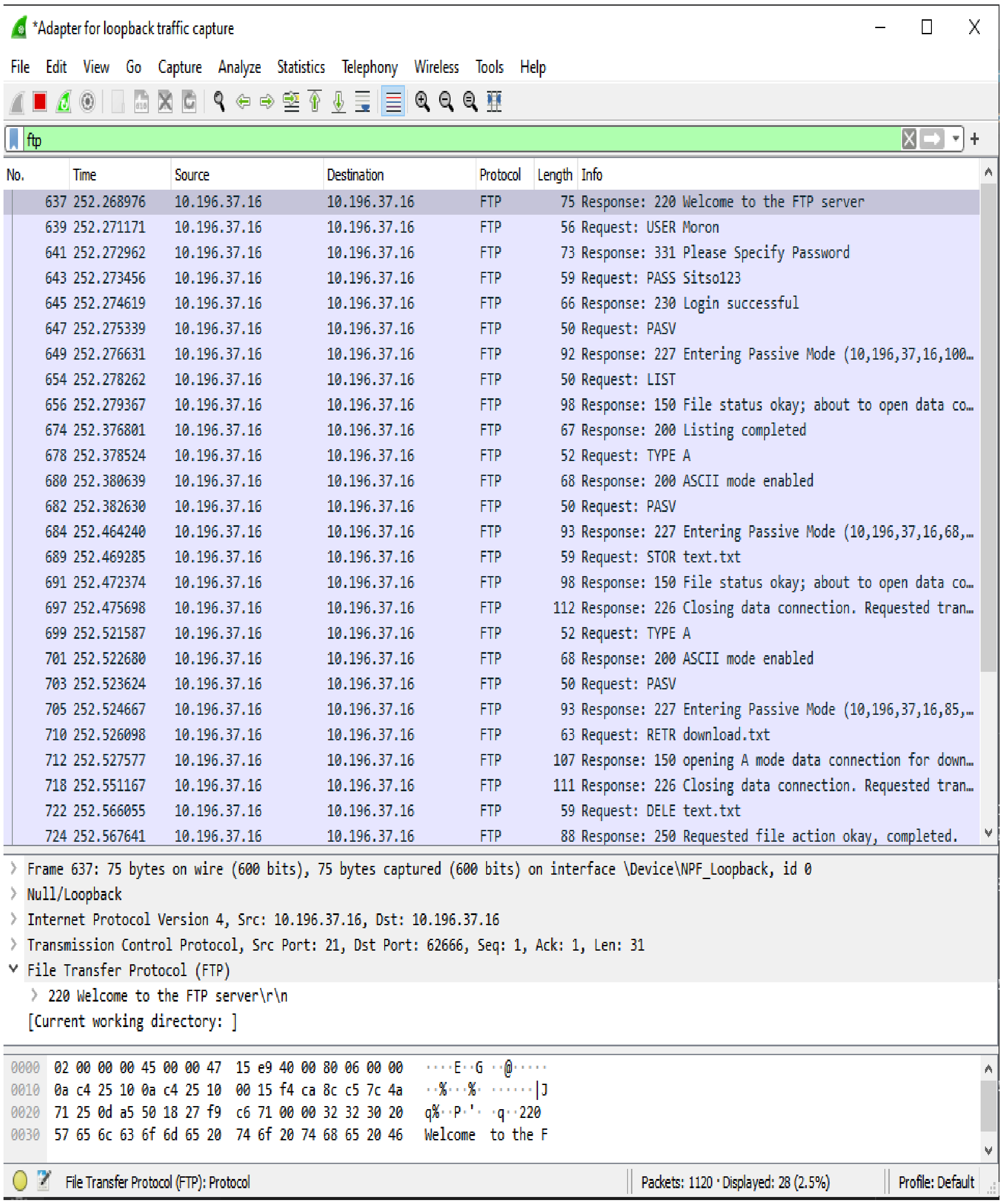
The server and client FTP application were tested and the results are presented. The captured packets on Wireshark provide evidence that the implementation of the FTP application is functional and it meets all the minimum requirements specified in the project brief. The design follows a good programming approach in order to make the code readable. The application would be easier to use with a user interface. In conclusion, the implementation of the FTP application was a success on the server side and it was not completed on the client time due to insufficient time to deign and implements the user interface.

References

[1] Imperva. “OSI Model.” https://www.imperva.com/learn/application-security/osi-model/, 2021. [Online; accessed 05-July-2021].
[2] J. P. abd J. Reynolds. “File Transfer Protocol.” https://datatracker.ietf.org/doc/html/rfc959, October 1985. [Online; accessed 01-July-2021].
[3] S. R. Technologies. “FTP – The File Transfer Protocol.”

APPENDICES

Appendix A: Results


*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ftp

No.  Time        Source        Destination   Protocol Length Info
 637 252.268976  10.196.37.16  10.196.37.16  FTP       75 Response: 220 Welcome to the FTP server
 639 252.271171  10.196.37.16  10.196.37.16  FTP       56 Request: USER Moron
 641 252.272962  10.196.37.16  10.196.37.16  FTP       73 Response: 331 Please Specify Password
 643 252.273456  10.196.37.16  10.196.37.16  FTP       59 Request: PASS Sitso123
 645 252.274619  10.196.37.16  10.196.37.16  FTP       66 Response: 230 Login successful
 647 252.275339  10.196.37.16  10.196.37.16  FTP       50 Request: PASV
 649 252.276631  10.196.37.16  10.196.37.16  FTP       92 Response: 227 Entering Passive Mode (10,196,37,16,100…
 654 252.278262  10.196.37.16  10.196.37.16  FTP       50 Request: LIST
 656 252.279367  10.196.37.16  10.196.37.16  FTP       98 Response: 150 File status okay; about to open data co…
 674 252.376801  10.196.37.16  10.196.37.16  FTP       67 Response: 200 Listing completed
 678 252.378524  10.196.37.16  10.196.37.16  FTP       52 Request: TYPE A
 680 252.380639  10.196.37.16  10.196.37.16  FTP       68 Response: 200 ASCII mode enabled
 682 252.382630  10.196.37.16  10.196.37.16  FTP       50 Request: PASV
 684 252.464240  10.196.37.16  10.196.37.16  FTP       93 Response: 227 Entering Passive Mode (10,196,37,16,68,…
 689 252.469285  10.196.37.16  10.196.37.16  FTP       59 Request: STOR text.txt
 691 252.472374  10.196.37.16  10.196.37.16  FTP       98 Response: 150 File status okay; about to open data co…
 697 252.475698  10.196.37.16  10.196.37.16  FTP      112 Response: 226 Closing data connection. Requested tran…
 699 252.521587  10.196.37.16  10.196.37.16  FTP       52 Request: TYPE A
 701 252.522680  10.196.37.16  10.196.37.16  FTP       68 Response: 200 ASCII mode enabled
 703 252.523624  10.196.37.16  10.196.37.16  FTP       50 Request: PASV
 705 252.524667  10.196.37.16  10.196.37.16  FTP       93 Response: 227 Entering Passive Mode (10,196,37,16,85,…
 710 252.526098  10.196.37.16  10.196.37.16  FTP       63 Request: RETR download.txt
 712 252.527577  10.196.37.16  10.196.37.16  FTP      107 Response: 150 opening A mode data connection for down…
 718 252.551167  10.196.37.16  10.196.37.16  FTP      111 Response: 226 Closing data connection. Requested tran…
 722 252.566055  10.196.37.16  10.196.37.16  FTP       59 Request: DELE text.txt
 724 252.567641  10.196.37.16  10.196.37.16  FTP       88 Response: 250 Requested file action okay, completed.

> Frame 637: 75 bytes on wire (600 bits), 75 bytes captured (600 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 10.196.37.16, Dst: 10.196.37.16
> Transmission Control Protocol, Src Port: 21, Dst Port: 62666, Seq: 1, Ack: 1, Len: 31
v File Transfer Protocol (FTP)
  > 220 Welcome to the FTP server\r\n
  [Current working directory: ]

0000  02 00 00 00 45 00 00 47  15 e9 40 00 80 06 00 00   ····E··G ··@·····
0010  0a c4 25 10 0a c4 25 10  00 15 f4 ca 8c c5 7c 4a   ··%···%· ······|J
0020  71 25 0d a5 50 18 27 f9  c6 71 00 00 32 32 30 20   q%··P·'· ·q··220
0030  57 65 6c 63 6f 6d 65 20  74 6f 20 74 68 65 20 46   Welcome  to the F

File Transfer Protocol (FTP): Protocol    Packets: 1120 · Displayed: 28 (2.5%)    Profile: Default
```

Fig. 1. Packets captured on wireshark