

Networks Assignment Report

We have been tasked with creating a python file sharing application with the aim of teaching us about basics of protocol design and socket programming for TCP connections. The brief provided to students specifies a client-server architecture and suggests features to add. This is a report summing up the functionality and protocol specification of said application.

Description of system functionality and features:

Setting up a connection



```

C:\windows\py.exe
Enter IP of server:
#
Attempting connection...
Attempting connection...
Attempting connection...
Attempting connection...
Attempting connection...
|

```

A client waiting for the server to start

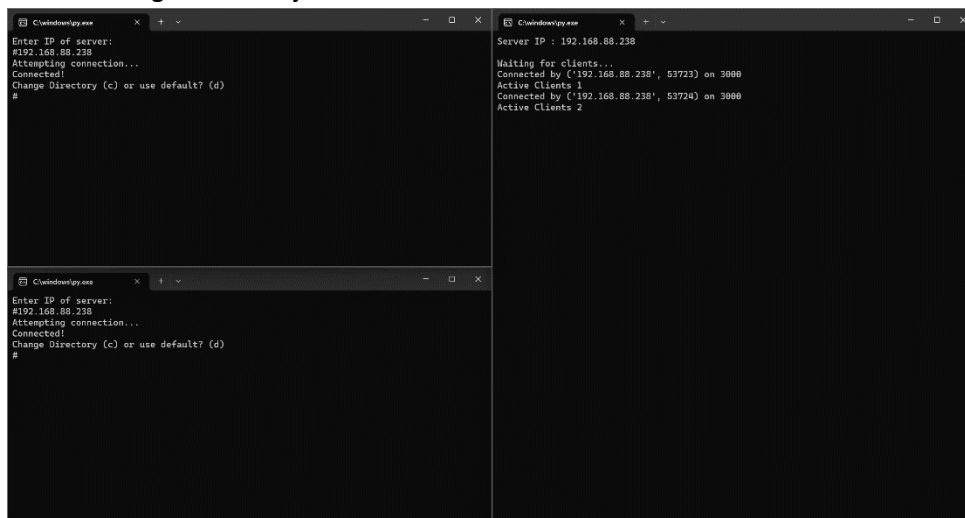


```

C:\windows\py.exe
Server IP : 192.168.88.238
Waiting for clients...
|

```

A server waiting for clients to join



```

C:\windows\py.exe
Enter IP of server:
192.168.88.238
Attempting connection...
Connected!
Change Directory (c) or use default? (d)
#

C:\windows\py.exe
Server IP : 192.168.88.238
Waiting for clients...
Connected by ('192.168.88.238', 53723) on 3000
Active Clients 1
Connected by ('192.168.88.238', 53724) on 3000
Active Clients 2

C:\windows\py.exe
Enter IP of server:
192.168.88.238
Attempting connection...
Connected!
Change Directory (c) or use default? (d)
#

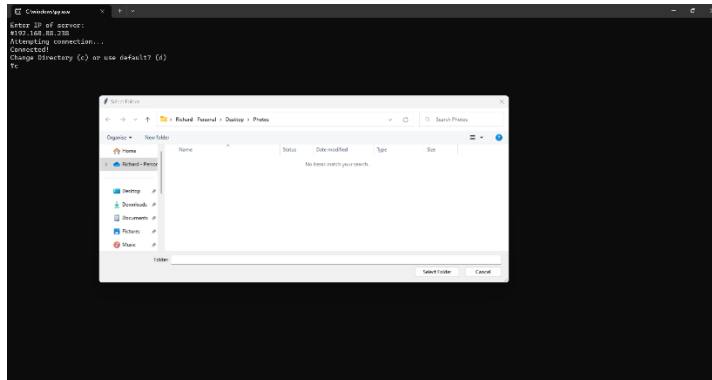
```

Here we can see the server has connected to 2 clients

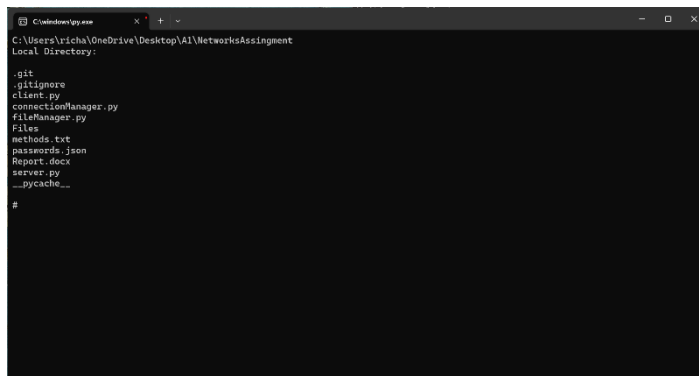
Server and Client Directories

Upon starting the client, the user will be prompted to where they would like their files to be stored. If they choose the default directory, all files the client downloads will be stored in the current working directory of the client. Otherwise, the client can specify which directory they would like for their

downloaded files to be stored. On the command line interface, the user has the ability to list all the files in their own directory and also list the files in the server directory (for convenience the screen is cleared between commands).



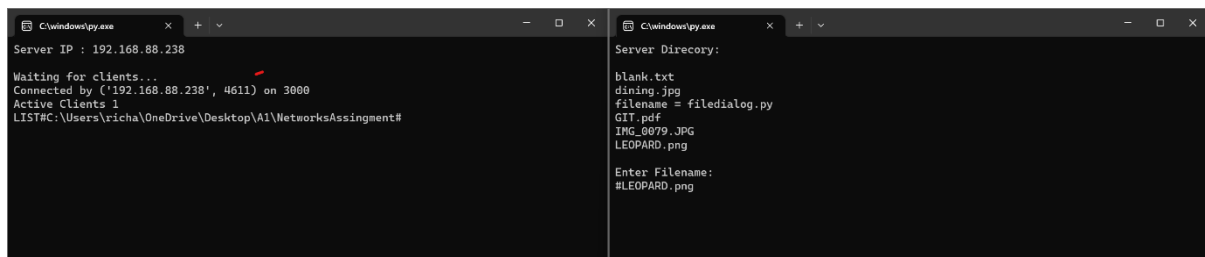
Client prompting the user to choose the window which they would like to make their directory.



Displaying client directory files

Uploading and downloading

Also using the command line interface, the user can both upload files to the server, and download files from the server. This functionality works just as you'd expect. However, as we know, waiting for files to download is never fun. To ease this burden, we have implemented a loading bar to show the progress of file uploads and downloads. We also added a feature to



The server and the client's response to the client requesting for the list of files on the server directory.

Note: If you are new to using this program, or forgot certain features, you can type "h" to view the help menu which displays all client commands, the help menu is show below

```

C:\windows\py.exe
HELP MENU
-----
Upload - (u)
Download - (d)
List Server Directory - (l)
List My Directory - (m)
Help - (h)
Change Directory - (c)
Quit - (q)
#

```

Help Menu

```

C:\windows\py.exe
C:/Users/richa/OneDrive/Desktop/LEOPARD.png
Successfully sent file C:/Users/richa/OneDrive/Desktop/LEOPARD.png
Success! ~ Checksum match
#

C:\windows\py.exe
55.8%
56.1%
57.2%
58.3%
59.4%
60.5%
61.6%
62.7%
63.8%
64.9%
66.0%
67.1%
68.2%
69.3%
70.4%
71.5%
72.6%
73.7%
74.8%
75.9%
77.0%
78.1%
79.2%
80.3%
81.4%
82.5%
83.6%
84.7%
85.8%
86.9%
88.0%
89.1%
90.2%
91.3%
92.4%
93.5%
94.6%
95.7%
96.8%
97.9%
99.0%
Files/LEOPARD.png
Checksum Match - File was not altered in transit

```

Client uploading file to server

```

C:\windows\py.exe
56.1%
57.2%
58.3%
59.4%
60.5%
61.6%
62.7%
63.8%
64.9%
66.0%
67.1%
68.2%
69.3%
70.4%
71.5%
72.6%
73.7%
74.8%
75.9%
77.0%
78.1%
79.2%
80.3%
81.4%
82.5%
83.6%
84.7%
85.8%
86.9%
88.0%
89.1%
90.2%
91.3%
92.4%
93.5%
94.6%
95.7%
96.8%
97.9%
99.0%

C:\windows\py.exe
59.4%
60.5%
61.6%
62.7%
63.8%
64.9%
66.0%
67.1%
68.2%
69.3%
70.4%
71.5%
72.6%
73.7%
74.8%
75.9%
77.0%
78.1%
79.2%
80.3%
81.4%
82.5%
83.6%
84.7%
85.8%
86.9%
88.0%
89.1%
90.2%
91.3%
92.4%
93.5%
94.6%
95.7%
96.8%
97.9%
99.0%
Files/LEOPARD.png
Checksum Match - File was not altered in transit
LISTING C:/Users/richa/OneDrive/Desktop/A1/NetworksAssignment#
c:\LEOPARD.png#
Files/LEOPARD.png
Successfully sent file LEOPARD.png

```

Client downloading file from server

Encryption

To protect the files as they are being sent between the client and server, this application has implemented end-to-end encryption. In *connection_manager.py* we have wrapped the socket in an encryption layer, adding encrypt and decrypt functions that makes use of the built in cryptography library called Fernet. Now if someone intercepts messages being sent between the client and the server, the

messages will be unreadable without knowing what the encryption key is, thus keeping users' files safe from prying eyes. The key can be specified by a server administrator.

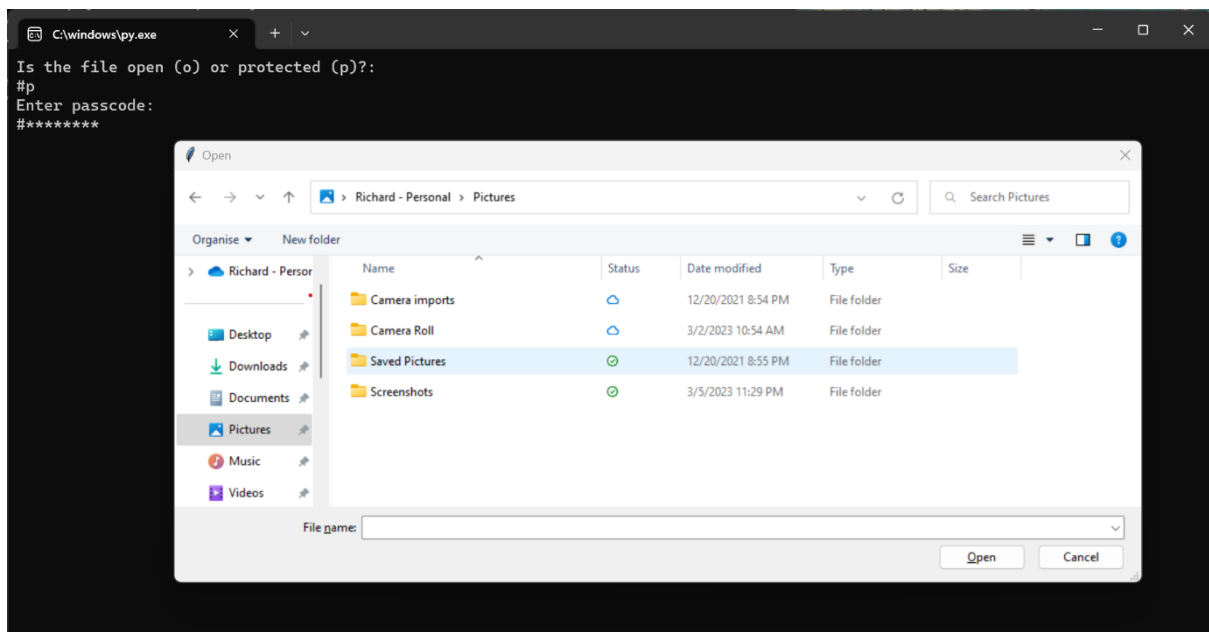
Checksum

This system includes a file validation system to check that the files sent are not altered in transit or corrupted.

This is achieved by using a md5 checksum generated in the send methods of both the server and the client. The checksum is then passed as an argument in the header and checked on the other side. In the event of the checksum not matching up, the corrupted or altered file is deleted.

File Authorisation

The user is prompted to indicate the file sharing permissions by indicating if the file is open (able to be downloaded and removed from the server by any user) or protected (only downloadable and removeable by users who have the secret passcode). If they choose to upload a protected file, they will be promoted to provide a passcode which will be added to a JSON file which will store all protected filenames and their corresponding passcode. If this user or another user wishes to download or remove this file they will be prompted to enter the file's passcode. If an incorrect passcode is provided the server goes back to waiting for a new command. If the correct passcode is provided the file proceeds to download to the client or be removed from the server (depending on what the command was).



Showcase of file protection

Protocol design & specification

As per the description brief, this application uses Client-Server architecture. Here the server contains all the files, and the clients connect to the server to upload and download files.

Reliable Data transfer

The program ensures when files are uploaded or download, they are sent reliably. This means that if the client loses connection midway through transferring a file, they will be able to reconnect and then resume the download of the file. This is achieved by chunking the file. When the connection drops the latest confirmed chunk on the client is recorded and then when the client reconnects it specifies which chunk it needs to start the download from again and continues with the download.

Message Header format

The format of the header message is simple, which allows efficient and fool proof communication between the client and server. The header command are: GET, LIST, POST, DELETE. Here is an example of a header message that will request to download a file:

GET#file.txt#

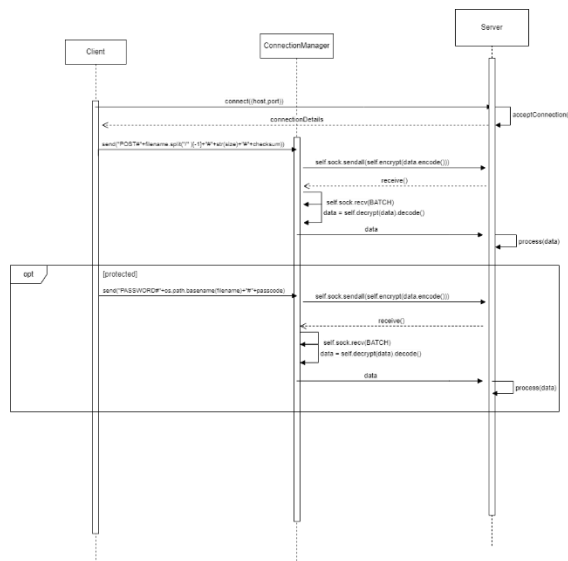
The command is followed by any number of parameters that can be used for various functions. The data transfer has a short header containing the size of the file to be transferred and then the checksum of the file. After that the file is then sent through. Control messages are simply messages that can occur during errors such as:

404 – File not Found

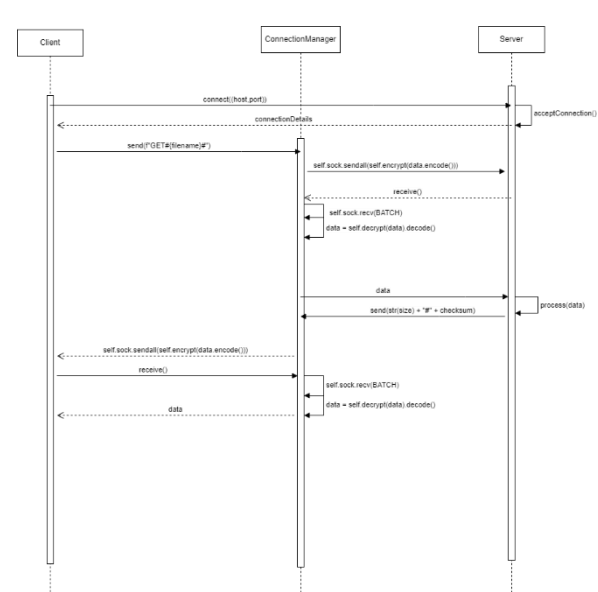
The basic rules or framework of communication between the client and server are as follows. Once connection has been established, the server waits for requests from the client. If the client sends a message through the server responds accordingly and then returns to waiting once complete. In the way the server and client are always taking turns.

Within the program the transport layer is handled by a single class file called *connectionManager*. It handles the sending of socket messages as well as the conversion of bytes to string messages and encryption/decryption by wrapping the send and receive in another method. It also handles connection errors, such as the connection dropping or aborting.

Upload Sequence diagram



Download Sequence Diagram



Robust protocol implementation (Stress Tests)

This program aims to use effective error catching to make sure the user cannot crash the client or server, being built on the principle of GIGO (garbage in, garbage out), not allowing the server to process anything unless the input matches very specific server commands. If the user inputs an unknown server command, an appropriate error is displayed, then they are prompted to input a new command or input. Because of this (and our effective protocol controlling server and client messages), no input from a client can crash the server, thus the server service is very reliable.

Many filetypes have been tested and various sizes (some of these examples have been uploaded with the project, look under the *files* folder) when uploading and downloading. We did encounter errors when trying to upload an image with hex values, but that was only when using a shift cypher to encrypt and decrypt messages. The encryption type was later changed.

We also tested many clients connecting to the server, we noted the server could easily host 10 clients, yet the theoretical number is much higher.

We also have various performance gaining features, such as the server killing off threads when the client disconnects.

Error Handling

As part of our design, we have implemented an extensive error checking list to help give the user useful information if a problem occurs (or if they have done something wrong) and to avoid unwanted crashes.

This list includes, but is not restricted to:

Error	Program action
Incorrect password on protected file	Deny user access to protected file
User inputs wrong file name when downloading	Discard input and ask for a new input
Checksum Mismatch	Delete corrupt file
User doesn't specify a file directory.	Use current working directory of the client
Unknown server command	Discard input and ask for a new input
Client loses connection	Notify Server