Following approach was taken to improve the efficiency and design of the project :

- For better Understanding, a structured approach has been used and there is a good Class design with suitable methods which separate the functionalities.

  Server Class : Server Implementation
  Client Class : Client Implementation
  AddressList Class: Custom linked list to store the ServerIP List and PeerLists
  Help Class : For displaying helps and usages

- Server and Client Classes, both have Initialize functions which does the basic initializations.

- The run function implements the loop and accepts the User Commands.

- The Server_IP list uses the AddressList Class which has a Linked List implementation and hence saves memory. It also has a serialize() and deserialize() functions which is able to convert an object to a char stream and reconstruct an object from char stream. It makes the transfer of the lists easy over the sockets.

```
int AddressList::serialize(char * stream )
int AddressList::deSerialize(char * stream)
```

- Also there is a significant improvement in comparisons in case of comparing the command.
  A usual style would have been an if else ladder.
  By using switch case a lot of comparisons can be saved.
  Eg.

```
switch (commandInterpretor(command_arg[0])) {
case HELP:
        help->displayUsageClient();
        break;

case CREATOR:
        std::cout << "Full Name : Sarang Dev" << std::endl;
        std::cout << "UBIT Name : sarangde" << std::endl;
        std::cout << "UB Email Address : sarangde@buffalo.edu"
                        << std::endl;
        break;

case DISPLAY:
```

  Also the commands have been used as Enums so that they are easy to identify.

- For many checks Booleans have been used to store the states.

  Eg. Bool registered has been used for many checks and was only set when successful registration occurs. Such variables helps in saving time as a check would require a traversal of the list.

Some Performance and Design Improvement which could be future enhancements:

1. Poll instead of Select
   The IO Multiplexing is achieved by the use of select() api which has certain demerits like :

- The FD_SET of descriptor is always modified by each select() call. It then has to be copied again.
- There is also a limit to the maximum of the sockets a select can monitor,i.e 1024.

The select can be replaced by poll and epoll/libevent which doesn't suffer from there demerits and have better performance.

2. Non Blocking Sockets
   The program uses Blocking sockets. So each time a recv() or accept() is called it blocks.
   To improve the performance, Non Blocking sockets can be used, as it doesn't wait for the recv() or accept() calls and can save the waiting time for the blocking calls.

3. Checksum for file
   The current implementation just checks if the file size is equal the file size in peers disk, to complete the transfer. But for safer implementation, a checksum can be done to ensure that the file has been transferred correctly.