Computer Networks- CEN 502

***Abstract*—In this project, a simple P2P and server network is designed that is used or transfer the text file from one host to another. The network makes use of both P2P as well as client server architecture to communicate and transfer the file.**

Keywords—Client, Server, P2P, Text file sharing.

I. INTRODUCTION

P2P systems evolved keeping in mind the egalitarian architecture of the internet that treated all the hosts in the network with equal importance. Examining the existing network architecture of P2P systems it can be concluded that there are mainly three types of P2P systems depending upon the presence of server, number of servers and the extent to which peers rely on these servers for communicating with other peers. Though there is no fine distinction between different types of P2P networks, broadly the networks can be divided into three main categories, they are 1) Centralized P2P network 2) Decentralized P2P network and 3) Hybrid P2P network that combines the features of first two network architecture. As already mentioned, the P2P system was designed keeping in mind the egalitarian architecture of the internet and so decentralized servers become important. There are various design issues that needs to be addressed particularly for the decentralized P2P network.

1) **Centralized P2P Systems:** These systems have the properties of both client-server as well as peer to peer system. These system have one or more servers that occasionally interact with clients for providing some information or get some information from the clients. Typically, servers are used to provide the clients with information about other clients who have certain data or block of data. A client queries the server for this information. Server provides the client with information. At this point the calling client stops interacting with server and can interact with other client directly for obtaining the file. Similarly, different clients periodically update their status to server when they have change in the block of data that they are storing. Server accordingly, changes its database i.e. information it is having about all the clients. Since Serer maintains the list of active clients in the network, an active client an exit the network by informing the server about it. Server updates its database and stops referencing that client in future update to clients. Since this type of network has properties of centralized client-server, it suffers from same attacks and typically same problems that centralized client-server has. Some of them are 1) Central server can be attacked with malicious software or virus. 2) The network can become very slow if there are many clients and too few servers to serve them. 3) This type of network is not scalable or robust. 4) It can be difficult initially for the network to take off the ground. 5) Single point of failure of Server is a biggest issue. Example of this network include famous music sharing site Naspers. Our project is also typical example of this type of network architecture as both server and client is involved.

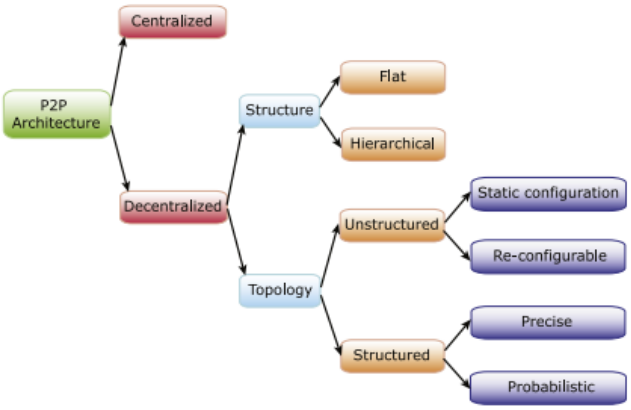

Figure 1: Hierarchy of Network Architecture

2) **Decentralized P2P Systems**: This type of network does not contain central server and only peers or clients are involved. In decentralized P2P system, the client does not see the entire network and has the information only of the partial network. Every peer has equal importance or weightage in the network. Each peer interacts with other peers to get the information of the data blocks present with other peers. As there is no central server involved the possibility of getting choked is less. Given no involvement of central server peer to peer network has some benefits over client- server network such as 1) It is more robust and stable than latter one 2) There is no single point of failure 3) The network is scalable and many clients can be connected in short period of time. Decentralized per to peer network can be broadly defined as flat (single tier) or hierarchical (multi-tier) systems depending upon how the functionality and load is distributed across the network. In a flat network the distribution is uniform. This offers simple solution but suffers from the drawback that it is difficult to debug the system or isolate the fault. This problem is addressed in hierarchical peer to peer system where fault detection is easier at the cost of more complex system. In a flat or unstructured P2P system, each peer is responsible for its own data. In addition to maintaining its own data it also maintains list of its neighbors and shares information with its neighbor only. There is no fixed rule or hierarchy as to which peer is responsible solely for a particular client. This poses some problem for this type of network which can be (a) it is difficult to predict which client has list of latest status of data and so finding the information about the data block can be difficult. (b) Entire network might need to be queried to get the complete answers. As querying only certain number of clients may result in partial answers and since querying client does not have information about which client has entire list so each and every answer needs to be queried. (c) The response might be delayed or might not come at all and it is difficult to pinpoint the reason for this. d) Even a query by one client can trigger heavy responses from all the neighboring and their neighboring clients. This heavy flow of information may congest the network. Example of this type of network is Free Net etc. One of the problem with decentralized P2P network is locating the peers or neighbors to a particular client. There are two options available with the client. It can either go for previously determined fixed number of neighbors or it can dynamically determine list of neighbors at each and every broadcast. The former is used and is based on the assumption that most of the time the list of retrieved neighbors will be same. This saves time and is efficient compared with dynamically determining the list of neighbors. This is however not always the case and latter approach needs to be implemented. This is in contrast with structured P2P network system, where DHT (Distributed Hash Table) is used. So a mapping between data and peers is used. These type of network use unicast-based lookup mechanisms to find resources. This is not efficient as far as response time is concerned and it might may be significantly delayed. It is however is efficient in terms of bandwidth. There is another type of decentralized P2P network which is exactly opposite to the one mentioned above in terms or

bandwidth utilization and response time. It uses broadcast mechanism basically, querying all the devices at the same time. It enjoys the benefit of low response time but high bandwidth as responses from all the clients floods the network.

On the contrary, in a structured P2P system, data placement is under the control of certain predefined strategies (generally a distributed hash table, or simply DHT). In other words, there is a mapping between data and peers.

In all these network generally actual data is used. Sometimes revealing actual data poses threat to privacy in which case metadata (data about data) is used. The network is fed with metadata and send across the network, once the actual data is required, data is generally encrypted and sent. At the receiving end data is encrypted using the public key of the sender which is generally sent along with the data. It is worthwhile mentioning that private key of the sender is used to encrypt the data.

3) **Hybrid P2P Systems**:

As already discussed, centralized and decentralized P2P have their own advantage and disadvantage. A centralized P2P network is used when there is no need for scalability and low response time is required. Further, server failure can be tolerated. This type of system are fast as there is low response time provided number of clients connected are not very large.

On the other hand if scalability and robustness is required, decentralized P2P is used. These offer immunity to single point of failure, scalability and robustness at the cost of low response time. It is possible to take the advantage of both the worlds by using the hybrid system that makes use of both the network. At first glance making a hybrid system might look difficult as the properties of centralized and decentralized system are contradictory to each other. This is achieved in the hybrid system by not keeping any centralized server, this ensures that network is decentralized. However to make the network centralized certain super nodes are chosen that act as peer as well server. These nodes are called super nodes as they serve both the purpose of central server as well as client. Querying of the neighbors can be done both using the peers as well as super nodes which act as central servers. The response time is generally low if the super nodes are used. By combining the features of peers and as well super nodes acting as central server, it is possible to increase the reliability, robustness and scalability of the system while at the same system lowering the response time.

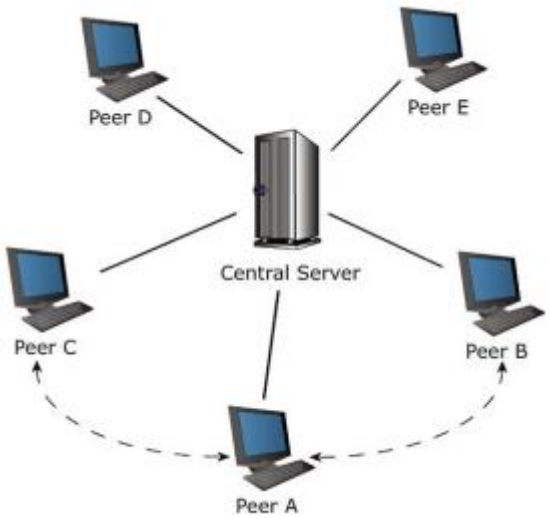

Figure 2: Client Server Model

II. COMPARISON OF DIFFERENT TYPES OF P2P NETWORK

Centralized P2P network:

- It has centralized server that is queried by clients. Clients seek information or update server with information.
- Chances of single point of failure of server.
- Low response time, not scalable or robust.
- Bandwidth is generally low compared to other type of the network.
- Ex. Naspers

Decentralized P2P network:

- It does not have centralized server and only clients query each other for network and data information.
- No chance of single point of failure
- High response time, scalable and robust.
- Bandwidth is highest compared to other type of networks.
- Ex. Free Net.

Hybrid P2P Network

- It does not have centralized server but has super nodes that acts as central server. It can be work as centralized as well as decentralized P2P network. As such it combines the best of both the networks.
- No chance of single point of failure. In case of super node failure, the network will continue to function as decentralized server.
- Low response time, as super nodes act as central server. The network is scalable and robust.
- Bandwidth can be controlled and is flexible.

III. FEATURES:

1) GUI Features:

The GUI is implemented using QT framework and C++ language. Only the GUI for peer client is implemented and the central directory server is console based. Following the basic parts of Peer client GUI.

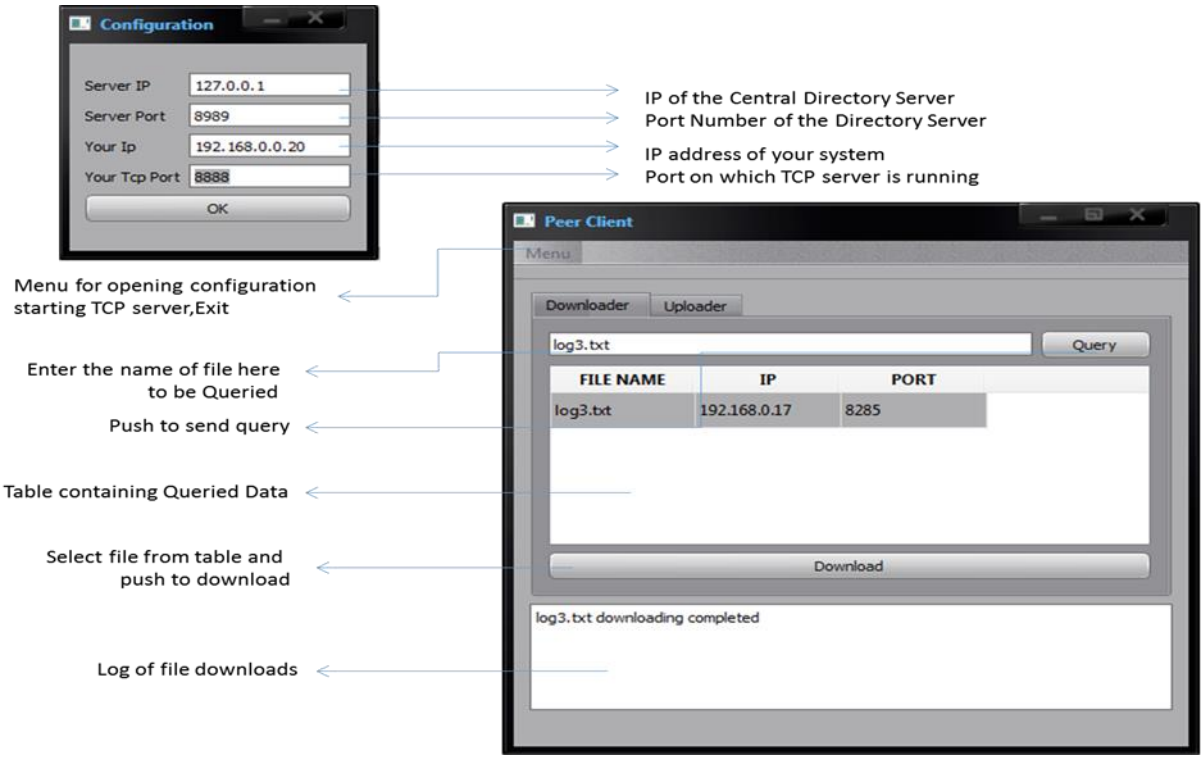

Figure 3: Configuration and Downloader

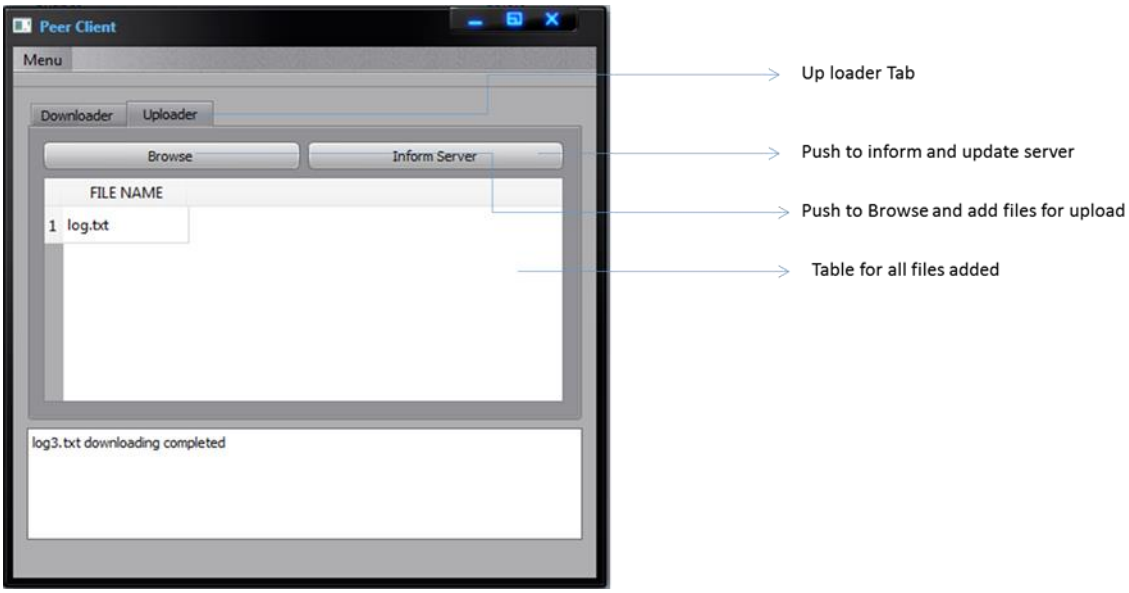

Figure 4: Uploader

Configuration Window: This is to configure the peer client mentioning the IP address and port for connecting to Central directory server, also we need to mention our own IP and port on which TCP server would run.

Downloader Tab: It provides the functionality for Query any file from the Central Directory Server, If the file is present with the central directory server then the downloader table gets updated with information regarding file (IP address and port of other peer client from where it can be downloaded).After the selecting the files and clicking on download button, file downloading starts and log window gets updated after the downloading process completes. The peer client can also simultaneously download multiple files by selecting on multiple files and clicking on download button.

Uploader Tab: It provides the functionality of adding files which client can upload to other clients on request basis. We browse file to add and the click on Inform server button to update the server about availability of files with you.

2) ***Salient Features:***

- Peer Client supports seamless multiple upload and downloads from other peer clients
- The timeout interval is updated at run time by updating sample Round trip time which provides better timeout time.
- The GUI is created using Model View architecture so it's faster and more responsive

3) ***Architecture:***

We have implemented a multithreaded peer client which cans act both as peer client and transient server. Also a central directory server which maintains directory of files from all the peer clients.
Following figure depicts the protocol between peer clients and directory server.

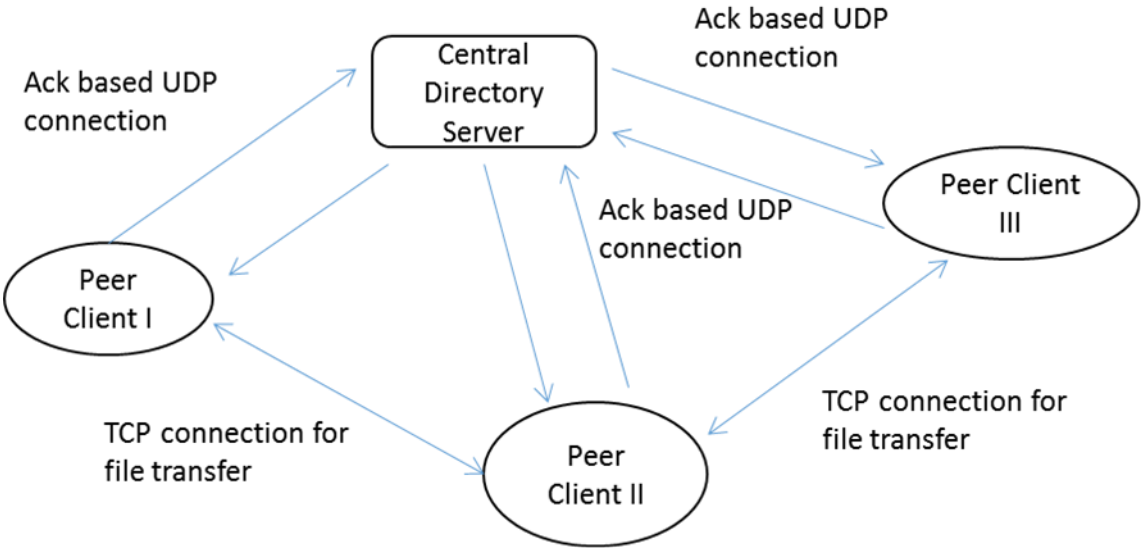

Figure 5: Architecture & Protocol

Protocol between Central Directory Server and Peer Client:

User Datagram protocol (UDP) is used for transferring packet between Central directory server and Peer client. Since UDP is not reliable acknowledgment is send between central directory server and peer client for every packet exchanged. After any of the system sends the packet it waits for particular time interval called timeout interval before retransmitting the packet. The timeout interval is calculated at real time by calculating the sample Round trip time of each packet received. The overall time out interval is calculated in the following way:

EstimatedRTT = (1 – α) * EstimatedRTT + α* SampleRTT
Where EstimatedRTT is taken as 100ms initially and α = 0.125
SampleRTT changes after every acknowledgement received by the system.

Then DevRTT is calculated, which is an estimate of how much SampleRTT typically deviates from EstimatedRTT:
DevRTT = (1 – β) *DevRTT + β *| SampleRTT – EstimatedRTT |.

Finally timeout interval is calculated using the following formula:

TimeoutInterval = EstimatedRTT + 4 •*DevRTT

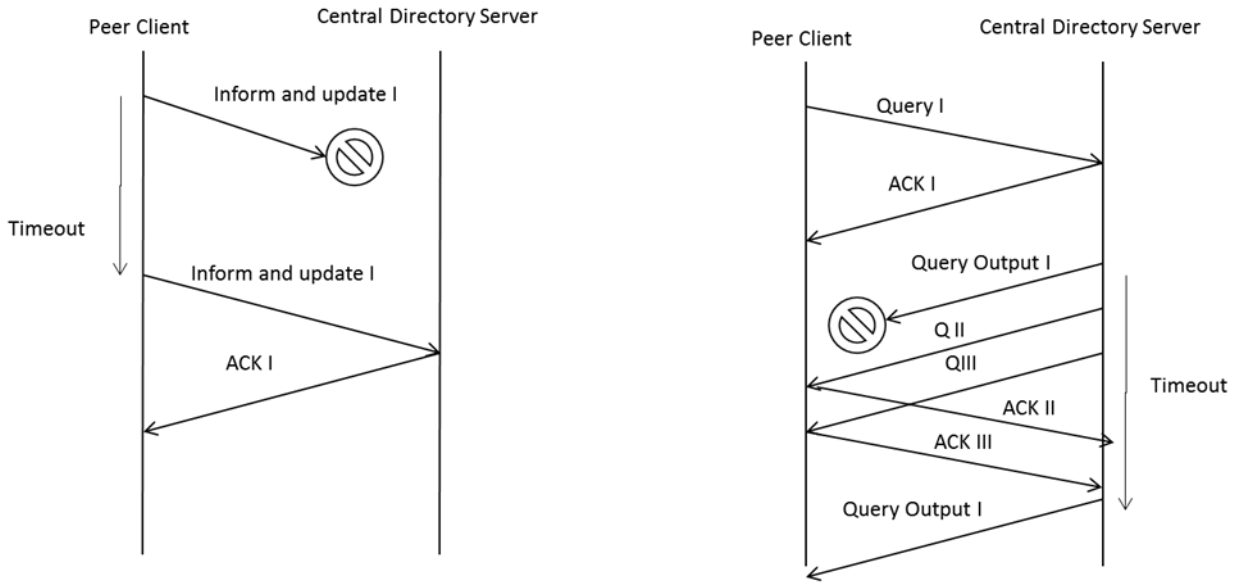

Figure 6: Protocol for communication between peer client and central server

4) ***Protocol between Transient Peer Server and Peer Client:***

Transmission control protocol (TCP) is used to create a reliable connection between both the systems. The transient peer server acts as a TCP server accepting multiple request from various peer clients from uploading and transfers the file .On the client side a spate download thread is created each time a file needs to be download. This tread connects to various transient server and accepts the files. Since MTU packet size is given as 128bytes so each file is fragmented into smaller packet size at sender side and then assembled at receiver side.

5) ***Implementation:***

Message Format:

For exchanging packets between peers and central directory server a particular message format has been defined.

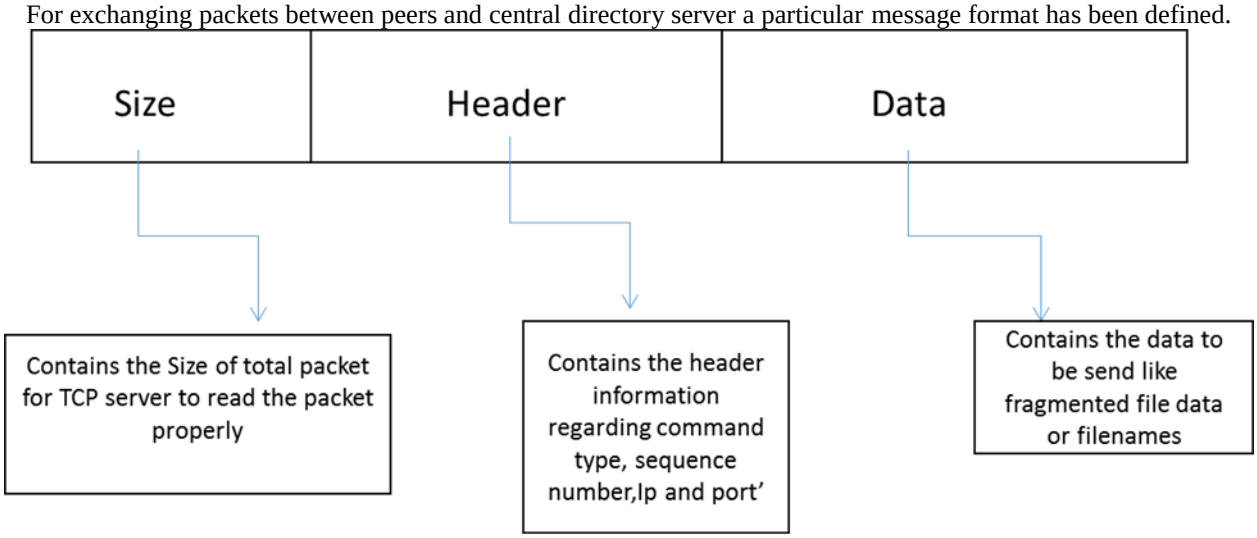

Figure 7: Message Format

Size is important as the TCP or UDP thread reading the packets from buffer reads the first 4 bytes containing the size of the packets and then reads the buffer according to size to get header and data packet.
Header field consist of header information like Command, sequence number, port and IP address of the sender etc.
Data field consist of the data which needs to be sent for example fragmented file data in case of file upload or filename in case of query for content.

6) **Message Serialization And Deserilaization:**

For creating packet we implemented various C++ classes and our own de-serialize and serialize functions to send the class objects over the network.
Following is an Example class:

```
Class ClientRequest
   {
   Private:
      STRING _fileName;
      STRING _userName;
      STRING _ip;
      SIGNED_INT _port;
    Public:
      ClientRequest () { }
      ClientRequest (char *buf);
      ClientRequest (STRING fileName, STRING userName, STRING ip, SIGNED_INT port)
      {
         _fileName = fileName;
         _userName = userName;
         _ip = ip;
         _port = port;
      }
      Int serialize (char *buf);
      STRING getFileName () {return _fileName ;}
      STRING getUserName () {return _userName; }
      STRING getIp () {return _ip ;}
      SIGNED_INT getPort () {return _port ;}
      Void setFileName (STRING fileName) {_fileName = fileName ;}
      Void setUserName (STRING userName) {_userName = userName ;}
      Void setIp (STRING ip) {_ip = ip ;}
      Void setPort (SIGNED_INT port) {_port = port ;}

      Void dump ();
   };
```

And Following Serialization and Deserialization typedefs (Example of char) are used to store data in class object in buffer for sending over the network

```
//for sterilization of char of 1 byte
#define SERIALIZE_8 (tmp, getMethod, buf, bytes) \
   tmp = getMethod; \
   memcpy (buf + bytes, &tmp, sizeof (tmp)); \
   Bytes += sizeof (tmp);


//for deserializing char
#define DESERIALIZE_8 (tmp, setMethod, buf, offset)\
   memcpy (&tmp, buf + offset, sizeof (tmp)); \
   SetMethod; \
   Offset += sizeof (tmp);
```

Similarly for integer, strings also above typedefs is created.
While serialization first we add the total size of packet and the command followed by data.
Commands and responses held us recognize the type of incoming message and act accordingly
Following commands and responses have been implemented:

```
Enum commands
   {
      INFORM_AND_UPDATE = 1, //for informing the server
      QUERY_FOR_CONTENT, //for sending query to server for a file
      ACK_CONTENT_FROM_SERVER,    //ACK for Query output
      CLIENT_REQUEST, //HTTP GET, Request other peer for file
      CLIENT_CONFIMATION_FOR_UPLOADING, //HTTP 400, transient peer sends                confirmation of
uploading
      CLIENT_FILE_NOT_FOUND, //HTTP 404, if file not available at transient peer server
      FRAGEMENT_FILE_DATA,    //for sending fragmented file data
      EXIT_BY_CLIENT //for exiting from directory server
    };
 Enum Response
      {
        QUERY_OUTPUT = 1, //response of query
        ACK_INFORM_AND_UPDATE_SERVER, //ACK for inform and update
      };
```

Command enum denotes set of commands send by peer to server or peer to transient server and peers,
Response enum denotes the response from central directory server to the peer clients.

IV. DESIGN AND IMPLEMENTATION OF CENTRAL DIRECTORY SERVER:

In directory server there is a single thread which reads all the data coming from all the peer clients.
The directory server mainly needs to handle two commands:

- `INFORM_AND_UPDATE`
- `QUERY_FOR_CONTENT`
- EXIT_BY_CLIENT

INFORM_AND_UPDATE command is send by peer client which wants to act as a peer server. It sends message including its own Ip address, filename it can upload to other peers and a port on which a TCP server is running and accepting requests for uploading a file.
Data Structure: We used a multihash having a string key as filename and value as a pair where pair. First is the Ip address and pair. Second is the port.

`Declaration:` QMultiHash <QString, QPair<QString, int> > _fileNameClientMap;

_fileNameClientMap is the main directory data structure of the central directory server. It contains all the updated filename listing and the respective Ip address and port where it is available.

QUERY_FOR_CONTENT command is send by peer client for querying for a particular file availability at other peer clients. The central directory server on receiving this command looks at central directory data structure _fileNameClientMap for particular file queried for and sends all the sender information regarding Ip and port of other peer client where it is available.

EXIT_BY_CLIENT command is send by peer client to directory server when it no longer want to server as transient peer server. In this case the central directory server removes all information of files from _ fileNameClientMap of that particular client.

After sending data the server issues a timer for each packet and checks after certain time called timeout for Acknowledgement of that particular packet. Acknowledgement and message are matched by sequence numbers. For this three central data structure are used.

```
QMap<QString, RttTimer*> _timerAckMap;
QMap<QString, RSP::QueryOutput> _clientData; //key is ip: sequence number
QList<QString> _ackClienMap; //key as seq: ip
```

_timerAck stores timers of each packet and stops them after Ack arrives

_clienData stores the packet sent, the timer threads takes data from this data structure and resends it if Acknowledgement is not recieved
_ackClienMap stores the incoming acknowledgements, the timer thread checks this data structure to find if Ack has been recieved or not based on peer client and sequence number

1) ***Pseudocode:***

```
CentralDirectoryServer ()
{
        Create and bind an udp socket udpSocket
        Create char buffer [128];
        While (read)
        {
                udpSocket.readfromBuffer (buffer, maxUdpSize);
                Command = buffer [4];
                switch (command)
                {
                Case INFORM_AND_UPDATE:
                Convert to object using deserialize and store data in central directory data structure;
                Break;
                Case QUERY_FOR_CONTENT  :
                {
                Send data to client about queried file;
                Issue timer for acknowledgment based on sequence number for  the packet send;
                Store packet in sent packet data structure
                }
                Case EXIT_BY_CLIENT:
                {
                Remove all information regarding this client from central directory data structure;
                }
                Case ACK:
                {
                        Stop timer of particular packet from timer data structure;
                }
}
```

2) ***Design and implementation of Peer Client:***

The peer client interacts with both the central directory server and other peer clients. It sends to query to central directory server for the file, for downloading it first send request to one of the other peer client based on the reply from central directory server. After receiving the confirmation it creates a downloader thread.
The main responsibility of downloader thread is to receive all the fragmented file data, reassemble it and then write it to a file. For reassembling in confirmation the transient directory server sends value of total number of packets of fragements files data .Also a number is added in each fragmented file data which denotes the order of data in file.
Thus downloaded thread creates a map of number and data:

```
QHash<int, QString> _dataMap;
```

Where all data is stored and then written to file sequence wise.
Similar to central directory server time out interval is also implemented here for checking acknowledgment for sent packets from central directory server.

3) ***Pseudocode of peer server:***

```
PeerServer ()
{
        Issue query to central directory server
                Receive query result;
        Issue a download request to other peer
```

```
                    Receive confirmation and initialize max packets to be recieved for a file;
            If request accepted issue a downloader thread Downloader
}
 threadDownloader
{
            Receive data using tcp socket

            Store data in _dataMap according to sequence;

            When all the packets recieved write the data to file in sequence;
}
```

4) ***Design and implementation of Peer Server:***

Peer Server is responsible for sending file requested by other peer clients. For this we created a tcp server ,when there is a new incoming connection it starts a thread sending it the socket descriptor .The thread issued reads the request and then sends the confirmation if file found to peer client who has requested the file.
Finally it starts sending the file by fragmenting it and attaching a sequence number so that it can be reassembled at the other end. Several such threads can be issued based on the incoming connection from various clients and thus can support multiple uploads;

5) ***Pseudo code for Peer server:***

```
PeerServer()
{
            Start a tcp server
            If new connection arrives start an uploader thread and send it the socket descriptor
}
uploaderThread()
{
            Start reading packet using socket descriptor
            Switch (command)
            {
                        Case file requested:
                                    Send confirmation
                                    Start sending the fragmented file data to the client
                                    End the thread;
            }
}
```

V. FUTURE SCOPE

Following things can be implemented to make this project more efficient

- Data encryption needs to be added to protect and secure data over network
- User interface can be greatly improved
- Ability to send multiple format files

REFERENCES

[1] Architecture of Peer to Peer network- Springer Science:

[2] Peer-Peer to network: Wikipedia.org

[3] Peer- to- Peer Architecture: https://tools.ietf.org/html/rfc5694