DESIGN OF P2P FILE SHARING SYSTEM

The P2P file sharing system consists of one centralized directory server and two peers. Each peer combines the functionality of a P2P client and a transient P2P server. As a P2P client, a peer makes requests to obtain information about available files from the directory server, and to obtain files from a peer serving files. As a transient P2P server, a peer serves files requested from a P2P client. The directory server, and each peer is uniquely identified by the name of its host computer and port number, where each host has a unique IP address

The programming language used is python as it is widely popular for socket programming and communication protocols in networks. The architecture used is like that of Napster.

Interaction between directory server and the P2P client

The communication between server and P2P client uses UDP sockets. The centralized directory server maintains a directory of text files that users are willing to share. It contains entries which list text file name, file size, and the host name where the file resides. The P2P client provides a text-based user interface for a user to interact with the directory server. With this interface, commands such as "Inform and update", "Query for content", and "Exit" are issued and responses returned from the directory server are displayed. There are three types of messages sent from P2P client to directory Server:

1. Inform and Update - For update of directory server. The directory server is multi-threaded and is able to handle concurrent writes to the directory of files. A user can send multiple entries in one message and end them with carriage return (CR) line feed (LF) characters. The directory server acknowledges the "Inform and update" message.

2. Query for content - When a P2P client wants to retrieve some content from its peers, it sends a "Query for content" message to the directory server specifying the file name or asking for a directory listing from the directory server. The client then follows the protocol for communication with a transient P2P server.

3. Exit - When a client wants to exit from this application, it sends an "Exit" message to the directory server. Upon receiving the message, the server deletes the entries associated with this client.

Protocol for implementing Server to Peer communication

Message Format

UDP sockets are used for communication between server and peer. For all communications the maximum transmission unit (MTU) is 128 bytes. We have two types of messages between server and the P2P client. They are request and response messages. The request message is send from P2P client to the directory server and a response message is sent from directory server to P2P client in response to client's request.

Request message

The different fields in the request message are:

1. Method - This field specifies the type of message used, whether it is Inform and Update, Query for content or Exit. Inform and Update is represented as "I", Query for content is represented as "Q", and Exit is represented as "E".

2. URL - This field specifies the client host name and port number.

3. Version - This field specifies the address of directory of files.

4. Sequence number - This field is used for reliable transfer of data. Each packet is assigned a sequence number to avoid duplicate packets in the receiver. Sequence numbers are randomly generated

5. Acknowledgment number - Used for reliable data transfer. Kept blank or unused in request messages.

6. Header field name – This field will indicate whether it is a file list or file name being updated or requested by the client.

7. Value field - It specifies the size of the body.

8. Body - This field contains the file list of the client in case of Inform and Update message and it contains the file name in the case of Query for Content message. It is kept unused in case of Exit message

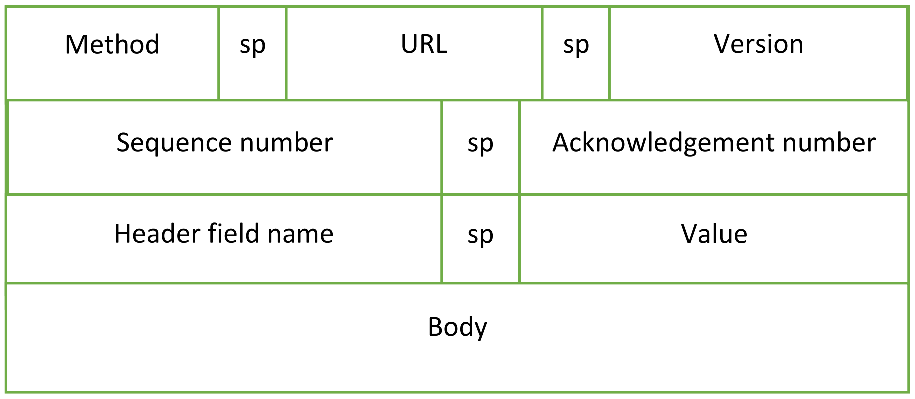


Figure 1. Format of request message

Response message

The different fields in the response message are:

1. Status code – Acknowledgment send in the form of 3- digit number. The status codes used are 200 and 404 and 400.

2. Phrase – Gives the explanation of the status codes. 200 means "OK" and 404 means that the server could not find what was being requested and 400 means bad request.

3. Sequence number - This field is used for reliable transfer of data. Each packet is assigned a sequence number to avoid duplicate packets in the receiver.

4. Acknowledgment number - Used for reliable transfer. To match the acknowledgment and the message for which the acknowledgment was issued.

5. Header field name – This field contains the result of the message.

6. Value field - It specifies the size of the requested file.

7. Body - It contains the IP address and port number of the client which has the requested file in case of Query for content message. In case of Inform and Update it will have the updated directory file list.

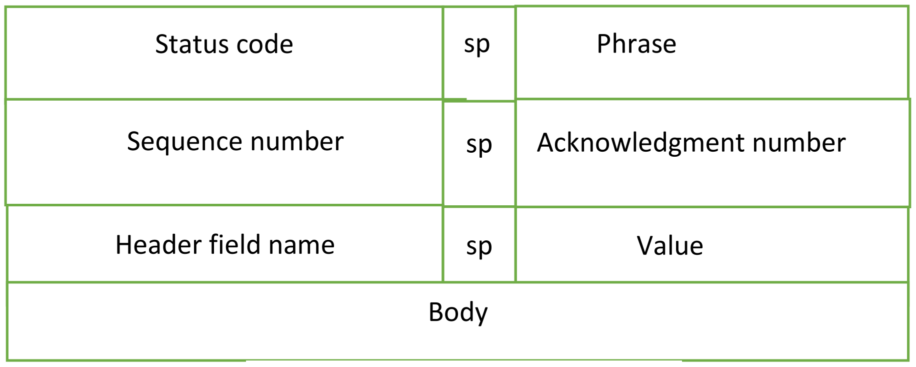


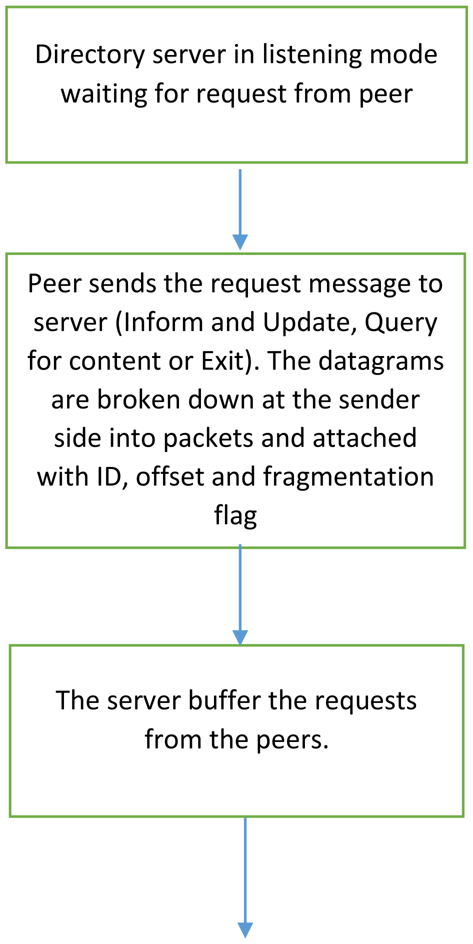
Figure 2. Format of response message

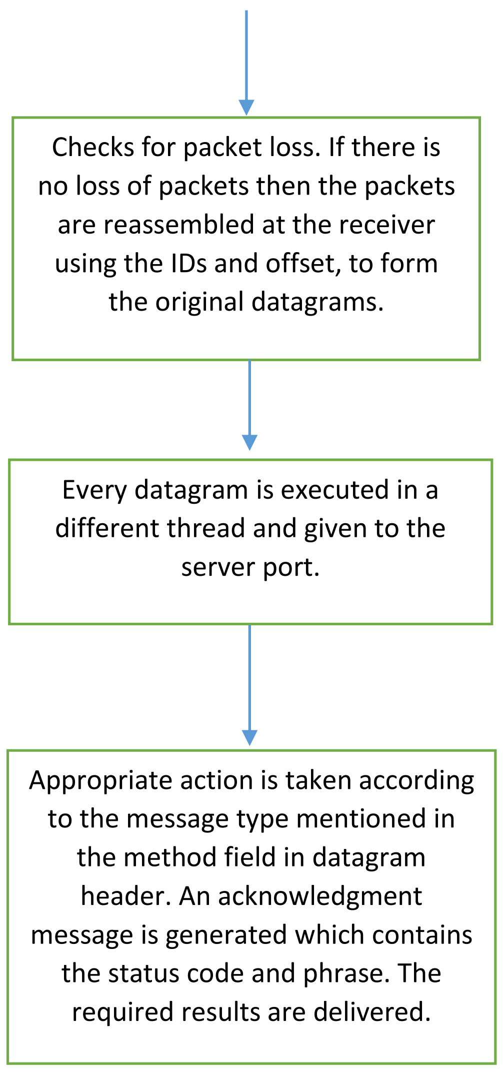
Maximum Transmission Unit (MTU) is 128 bytes. Huge chunks of data have to be fragmented into packets of 128 bytes and reassembled back at the server. While coding in python 20 bytes were lost during the process. So the body size of each packets is 96 bytes instead of 128 bytes. Each datagram is fragmented and header is added to each packet obtained. Header consists of fragmentation flag, length, Identification number (ID) and offset fields. Length specifies the total length of each packet. ID is same for all packets of a single datagram so that when the destination receives a series of packets from the same sending host, it can examine the identification numbers of the packets to determine which of the packets actually belong to the larger datagram. Because IP is an unreliable service, one or more of the packets may never arrive at the destination. For this reason, in order for the destination host to be absolutely sure it has received the last packet of the original datagram, the last packet has a fragmentation flag bit set to 0, whereas all the other packets have this flag bit set to 1. Also, in order for the destination host to determine whether a packet is missing (and also to be able to reassemble the packets in their proper order), the offset field is used to specify where the packets fits within the original datagram.

At the receiver side we first check for packet loss after a timeout period. The offset is extracted and the number of packets is obtained by dividing the offset by 12 (as body size is 96 bytes) and adding 1 to it. If this result is equal to the number of packets actually received, then there is no packet loss. If this result is less than the number of packets received, then there is packet loss and the entire datagram is discarded. The server continues the process with other packets, if any, or it will go into waiting stage again.

PROCESS FLOW BETWEEN DIRECTORY SERVER AND PEER

The flow can be summarized as below:


Directory server in listening mode waiting for request from peer
Peer sends the request message to server (Inform and Update, Query for content or Exit). The datagrams are broken down at the sender side into packets and attached with ID, offset and fragmentation flag
The server buffer the requests from the peers.


Checks for packet loss. If there is no loss of packets then the packets are reassembled at the receiver using the IDs and offset, to form the original datagrams.
Every datagram is executed in a different thread and given to the server port.
Appropriate action is taken according to the message type mentioned in the method field in datagram header. An acknowledgment message is generated which contains the status code and phrase. The required results are delivered.

Data Structures used in the design

There are two types of lists maintained in the Directory Server. A peer list which contains the list of peer IP addresses and the port numbers. And a full list which contains the filename also along with the IP addresses and the port numbers. The data structure used for both the lists are dictionary. The keys for peer list are Host and port and the corresponding values are IP address and the port number. Similarly for the full list there are three keys. They are host, port and filename, where the corresponding values are IP address, port number and filename. For all other purposes, the data structure list is used.

Reliability of Data transfer between directory server and the P2P client

Reliable data transfer is highly important in communication. In a network data can get corrupted or lost. We need methods to handle these scenarios. In order to implement reliability we used acknowledgements and time-outs for individual packets. Each message has a sequence number and an acknowledgment number. The sequence number and acknowledgment numbers are compared to find out which ACK message corresponds to which datagram. If a datagram is lost, the sender eventually times out and retransmits the datagram. If an acknowledgment packet is lost, the sender of the data still times out and retransmits the datagram. In this case the receiver of the datagram notes from the block sequence number in the datagram that it is a duplicate, so it ignores the duplicate of the datagram and retransmits the acknowledgement. If both the sender and receiver use a timeout with retransmission, and both retransmit whatever they last sent, a condition termed the sorcerer's apprentice syndrome results. When a time-out occurs every datagram and acknowledgement will be sent twice. So the sender never retransmit a datagram if it receives a duplicate acknowledgement. In order to determine the time-out interval we used the round-trip time estimation procedure outlined in textbook. We took estimated RTT as 100 ms.

The sample RTT, denoted SampleRTT, for a datagram is the amount of time between when the datagram is sent (that is, passed to IP) and when an acknowledgment for the datagram is received. The SampleRTT is being estimated for only one of the transmitted but currently unacknowledged segments, leading to a new value of SampleRTT approximately once every RTT. SampleRTT is never computed for a datagram that has been retransmitted; it only measures SampleRTT for datagram that have been transmitted once. In order to estimate a typical RTT, we calculate the average of the SampleRTT values called as EstimatedRTT, of the SampleRTT values. Upon obtaining a new SampleRTT, EstimatedRTT is updated according to the following formula:

$\text{EstimatedRTT} = (1 - \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}$

The value of $\alpha = 0.125$, so the formula becomes:

$\text{EstimatedRTT} = 0.875 \cdot \text{EstimatedRTT} + 0.125 \cdot \text{SampleRTT}$

Note that EstimatedRTT is a weighted average of the SampleRTT values.

Such an average is called an exponential weighted moving average (EWMA). The word “exponential” appears in EWMA because the weight of a given SampleRTT decays exponentially fast as the updates proceed. In addition to having an estimate of the RTT, it is also valuable to have a measure of the variability of the RTT defined as DevRTT, which is an estimate of how much SampleRTT typically deviates from EstimatedRTT:

$\text{DevRTT} = (1 - \beta) \cdot \text{DevRTT} + \beta \cdot | \text{SampleRTT} - \text{EstimatedRTT} |$

DevRTT is an EWMAof the difference between SampleRTT and EstimatedRTT. The value of β used is 0.25.

Given values of EstimatedRTT and DevRTT, the time out interval is calculated as:

$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \cdot \text{DevRTT}$

As soon as a segment is received and EstimatedRTT is updated, the TimeoutInterval is again computed using the formula above.

Pseudocode describing directory server to peer communication:

```
While (True)
    Server waits for request message from peer
    Peer fragments the message, attach header and sent it to server, starts timeout for server's
    response.
    Server receives the first packet
    Starts time out for more packets
    If (new packet received) resets the timeout
    After time out
    Checks for fragment loss (offset, ID, fragmentation flag)
                Find the fragment with same ID, get the item number
```

```
            Find the fragment with flag=1, get the offset from this fragment
            Fragment number= (offset/12) + 1
            If (Fragment number = item number)
                    There is no loss
                    Reassemble these fragments with same IDs into original message
                    Start a new thread for each reassembled message
                    Extract method data from each packet
                            If (method = E)      E for Exit
                                    Delete the information in peer_list and full_list
                                    which matches  port number and IP address
                            If (method = I)      I for Inform and Update
                                  Update the peer file list to the full list which
                                  Peer list stores IP address and port number
                                  Full list stores IP address , port number and file
                                  Response message sent
                                  (including only file list in the body)
                           If (method = Q)     Q for Query for content
                                  Search full list and respond back with IP address
                                  and port number of peer which has the file
                           else
                                    400  :  Bad Request
          Else if (Fragment number < item number):
                           Discard the packet
                           Continue the process with other packets or go to waiting
                           stage
 If (received acknowledgment):
          Jump to Peer to peer communication
If (not received acknowledgment)
          After peer timeout retransmit the packet
```

Interaction between P2P server and the P2P client

A P2P client can be a client of both the directory server and the P2P server. The P2P client communicates with the directory server first. Directory Server provides the hostname and address of the peer which has the required file. Client then selects a server from a list of P2P servers that have a desired file, and issue the HTTP GET command to retrieve desired text files from the selected P2P server. HTTP 1.1 is used to transmit text files between a P2P client and a P2P server. Separate HTTP requests are sent for each text file desired. P2P can handle 3 multiple simultaneous requests in parallel as it is multi-threaded. In the main thread, the server listens to a fixed port. When it receives a TCP connection request, it creates a new TCP socket and services the request in a separate thread.

The messages exchanged between P2P client and P2P server are:

1. HTTP GET – Issued by P2P client for requesting file from the P2P server.

2. Status codes- Issued by P2P server as response to request message from client. Code 200 indicates OK, 400 indicates Bad request, 404 indicates server could not find what was requested and 505 indicates wrong http version.

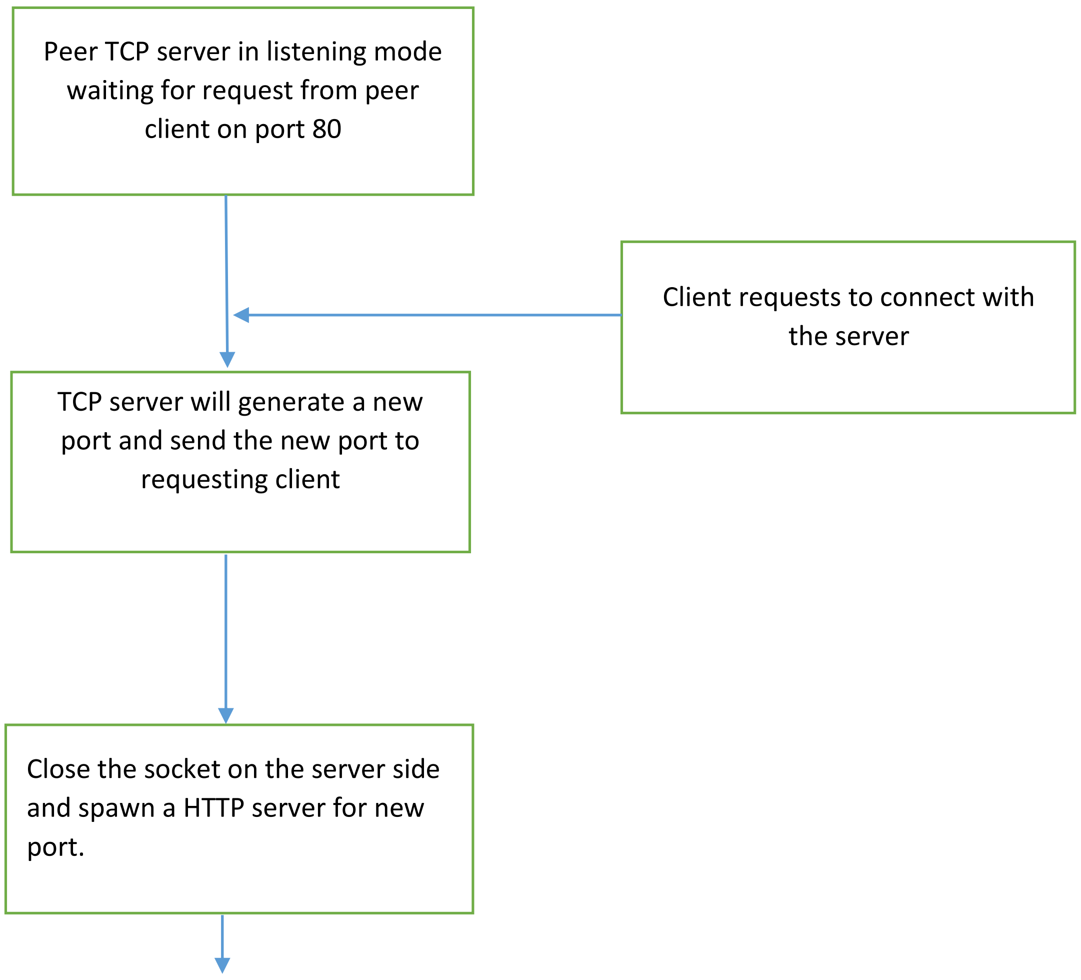
Protocol for implementing Peer server to Peer Client communication

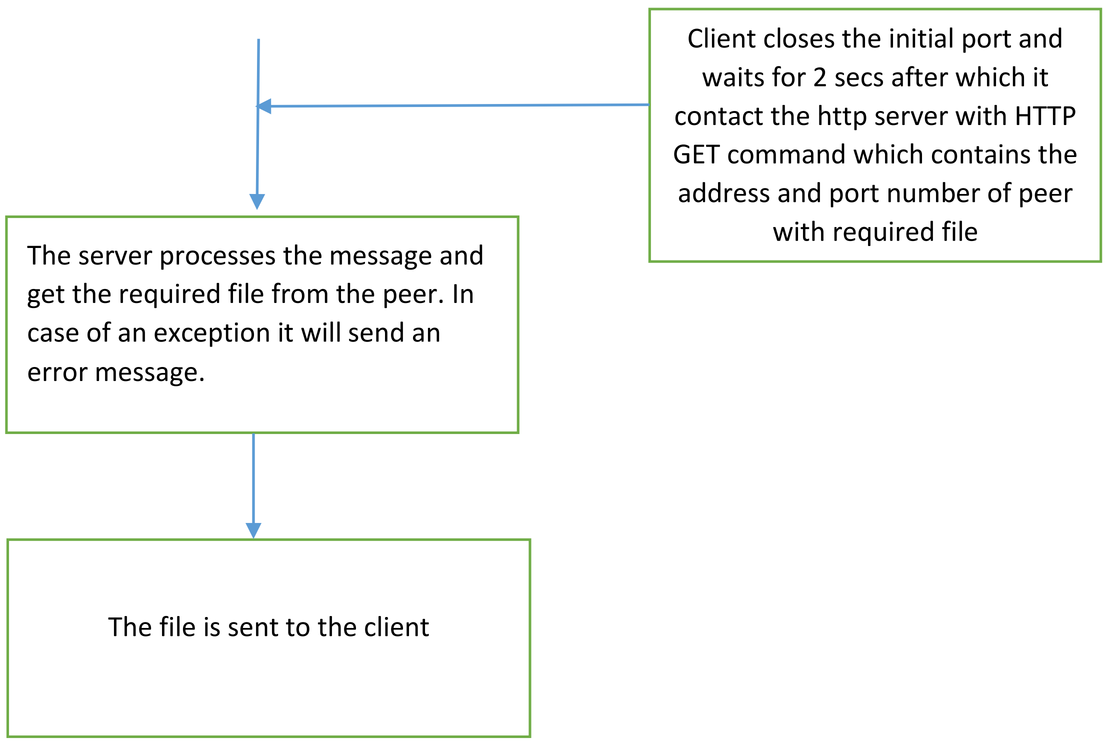
Communication between Peer server and peer client uses TCP sockets. Peer acting as server would be a TCP server which opens a connection socket for peer client through which it sends the port number for a new data socket. An HTTP server is spawned for this new port with the help of simpleHTTPserver library in python. The simpleHTTPServer module defines a single class, SimpleHTTPRequestHandler. This class serves files from the current directory and below, directly mapping the directory structure to HTTP requests.

PROCESS FLOW BETWEEN PEER SERVER AND PEER CLIENT

The flow can be summarized as below:

`


Peer TCP server in listening mode waiting for request from peer client on port 80
Client requests to connect with the server
TCP server will generate a new port and send the new port to requesting client
Close the socket on the server side and spawn a HTTP server for new port.


Client closes the initial port and waits for 2 secs after which it contact the http server with HTTP GET command which contains the address and port number of peer with required file
The server processes the message and get the required file from the peer. In case of an exception it will send an error message.
The file is sent to the client

Pseudocode describing peer server to peer client communication:

```
while(1):
      Peer TCP server.listen
      Client contacts with connect message through port 80(connection socket) on server
      Server sends the port number of a new data socket to client
      Close.connectionsocket
      Simple HTTP server socket created for new port number
Client. Wait ( 2 secs)
Contact with HTTP GET message
If( error):
              Return error status code
If(not error);
             Return status code "200"
             Process message and retrieve the file from the peer
Send the file to the client
```

Multi – threading capabilities

Directory Server and P2P client - The server is multithreaded to obtain requests from multiple users at a time. Each client will sent their request messages and each message is broken down into packets. As we mentioned, packets belonging to the same datagram will have the same IDs. So the packets are sorted according to their IDs and packets with the same ID are given to a single thread. So packets are reassembled to datagram and each group is assigned to a unique thread. The server side is robust as it can handle loss of packets and create separate threads for each client.

P2P server and P2P client – Whenever a new client comes in a new socket is opened for it and connection is made in a new thread. The peer server is TCP server and it opens a new thread for each client.

Pseudocode explaining the directory server

```
While (True)
    Server waits for request message from peer
    Peer contacts the server
    Server receives the first packet
    Starts time out for more packets
    If (new packet received) resets the timeout
    After time out
    Checks for fragment loss (offset, ID, fragmentation flag)
                Find the fragment with same ID, get the item number
                Find the fragment with flag=1, get the offset from this fragment
                Fragment number= (offset/12) + 1
                If (Fragment number = item number)
                     There is no loss
                     Reassemble these fragments with same IDs into original message
                     Start a new thread for each reassembled message
                     Extract method data from each packet
                             If (method = E)      E for Exit
                                 Delete the information in peer_list and full_list
                                 which matches  port number and IP address
                             If (method = I)      I for Inform and Update
                                 Update the peer file list to the full list which
                                 Peer list stores IP address and port number
                                 Full list stores IP address , port number and file
                                 Response message sent
```

```
                                                (including only file list in the body)
                                        If (method = Q)     Q for Query for content
                                                Search full list and respond back with IP address
                                                and port number of peer which has the file
                                        else
                                                  400  :  Bad Request
                        Else if (Fragment number < item number):
                                        Discard the packet
                                        Continue the process with other packets or go to waiting
                                        stage
```

Pseudocode describing the P2P client

```
if ( P2P client connects with directory server)
        Send the message
        Break datagram into packets and attach header
        Wait for ACK
          If (received ACK):
            If (message = Inform and Update)
               If (ACK = OK)
                      View the file list sent by the server
               If (ACK = error)
                      Go back to the main menu
           Else if (message= Query for Update)
               If (ACK= OK)
                       Retrieve the IP address and port number of the peer that contains the requested file
               If (ACK = error)
                      Go back to the main menu
           Else if (message= Exit)
             If (ACK= OK)
                       Disconnects from the system
             If (ACK = error)
                      Go back to the main menu
```

```
    If (NOT ACK)
                After peer timeout retransmit the packet

If (P2P client connects with P2P server)
        Sends connect request
        Receives data port number and IP address
        Waits for 2 secs
        Sends HTTP GET message with the file name specified in it
       If (ACK = OK)
           Retrieves the file
       If (ACK = error)
           Go back to main menu
```

Pseudocode describing the P2P server

```
while(1):
   Peer TCP server.listen
   Client contacts with connect message through port 80 on server
   Sends the port number of a new data socket to client
   Close.connectionsocket
   Simple HTTP server socket created for new port number
Receive.HTTP GET message
If( error):
       Return error status code
If(not error);
      Return status code "200"
      Process message and retrieve the file from the peer
Send the file to the client
```

Conclusion

The p2p file sharing system is designed in a robust way. It can handle packet loss by means of timeout. There can be clients that advertise the same file. In this case the address and port numbers of all the peers which has the same file is retrieved and sent to the client.

The P2P client can compare the retrieved file with files in the directory file list. The file retrieved from server is named as temp.txt. Then it will compare temp.txt with the files in the directory file list. If there is same file, it will check for the content. If the contents are same it will delete the temp.txt file. If the contents are different it will rename the temp file.

It can handle the query for content of various files at the same time. IP address and port numbers of all the peers will be displayed. The directory server and the P2P server are multithreaded.

References

- Computer Networking a Top Down Approach 6th Edition by James F. Kurose, Keith W. Ross
- https://docs.python.org/2/library/simplehttpserver.html
- https://www.python.org/
- https://en.wikipedia.org/wiki/Transmission_Control_Protocol
- https://en.wikipedia.org/wiki/User_Datagram_Protocol
- http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html
- http://www.tcpipguide.com/free/t_HTTPRequestMessageFormat.htm