CSE513 Peer-to-Peer File Sharing

-Abhishek Kumar(azk6085), Abhijeet Kumar(amk7371)

System Description

We designed a p2p file sharing system using TCP/IP in python. In our file sharing system we split the file into a chunk size of 64KB. We wanted our file sharing system to be user friendly thus used UI instead of having a clumsy command line or complicated json files. For demonstration we tried our system with 3 peer nodes and 1 central server, sharing 2 files with each other (1.txt - 36Bytes, 2.jpg - 130KB). Further in the writeup, we briefly describe the functioning of our system.

Usage:-

```
Python3 src/main.py [-h] -m MODE [-s SERVER_IP] [-p PORT_ID]
```

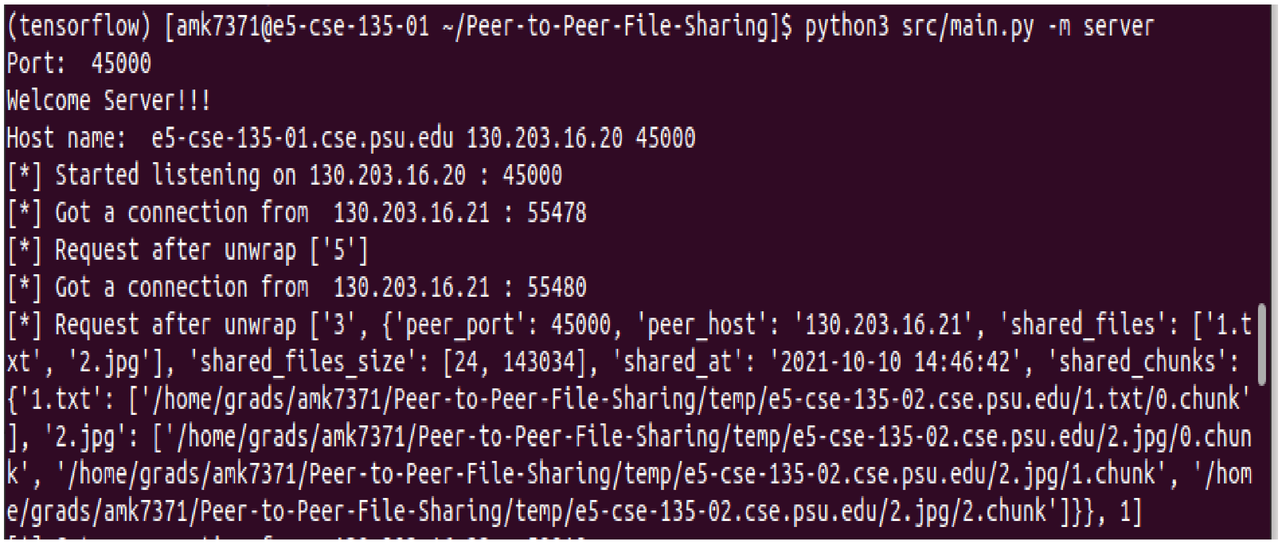
1. Both server and Peers can communicate with each other (send & listen) using TCP protocol.
2. Server internally maintains the list of files shared with the network.
3. Upon the receival of file register request by the server, it saves the file meta data and upon receival of file list message it shares the metadata with the requesting peer.
4. When peer request file location to the server, the server look into the metadata and share the position of each chunk.
5. Upon the download of each chunk, the new peer register itself as well as the source of the chunk by sending CHUNK_REGISTER command to the central server.
6. We designed peer in such a way that it supports multithreading with one daemon thread as listener and other writer thread which connects with different peer (max connection 5).
7. Peer upon receiving any data/message from the other peers or server display the result In terminal.
8. Peer uses rarest first scheme to decide the ordering of the chunks to be downloaded. It gets the metadata information from the central server and further dispatches the download thread based on the scheme. If all peers are available in equal quantity it goes with the lowest number first to decide the dispatching order. Peer downloads each and every chunk and store them inside a newly created directory called temp. Once all the chunks are downloaded it combines all the chunk and create final output file inside download directory.
9. If in case the peer crashes in between, it makes use of temp directory to understand what all chunks are already downloaded and then further download the remaining ones.
10. If in case the source peer leaves or crashes, destination peer uses pops the address from chunkid_to_addresses map and downloads from the next available one.
11. In our system we implemented choking/unchoking by storing 4 state variables {am_interested, am_choking, peer_interested, peer_choking} for each connection. A block is downloaded by the client when the client is interested in a peer, and that peer is not choking the client. A block is uploaded by a client when the client is not choking a peer, and that peer is interested in the client.

Protocol specification

1. Register Request: Each peer registers its address (string), port (int), count of files (int), and list of file path (string), number of bytes (int), and list of chunks where each item has chunkid (int)

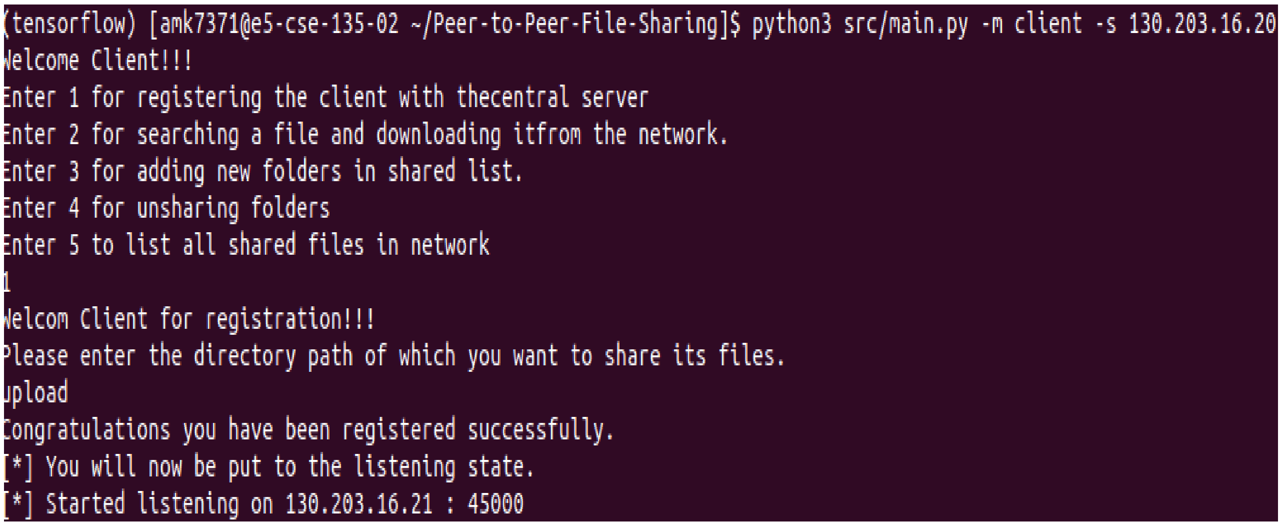
Server Response: status (bool) to demonstrates if the append was successful or not along with its entry id (int).

At central server -



```
(tensorflow) [amk7371@e5-cse-135-01 ~/Peer-to-Peer-File-Sharing]$ python3 src/main.py -m server
Port:  45000
Welcome Server!!!
Host name:  e5-cse-135-01.cse.psu.edu 130.203.16.20 45000
[*] Started listening on 130.203.16.20 : 45000
[*] Got a connection from  130.203.16.21 : 55478
[*] Request after unwrap ['5']
[*] Got a connection from  130.203.16.21 : 55480
[*] Request after unwrap ['3', {'peer_port': 45000, 'peer_host': '130.203.16.21', 'shared_files': ['1.t
xt', '2.jpg'], 'shared_files_size': [24, 143034], 'shared_at': '2021-10-10 14:46:42', 'shared_chunks':
{'1.txt': ['/home/grads/amk7371/Peer-to-Peer-File-Sharing/temp/e5-cse-135-02.cse.psu.edu/1.txt/0.chunk'
], '2.jpg': ['/home/grads/amk7371/Peer-to-Peer-File-Sharing/temp/e5-cse-135-02.cse.psu.edu/2.jpg/0.chun
k', '/home/grads/amk7371/Peer-to-Peer-File-Sharing/temp/e5-cse-135-02.cse.psu.edu/2.jpg/1.chunk', '/hom
e/grads/amk7371/Peer-to-Peer-File-Sharing/temp/e5-cse-135-02.cse.psu.edu/2.jpg/2.chunk']}}, 1]
```

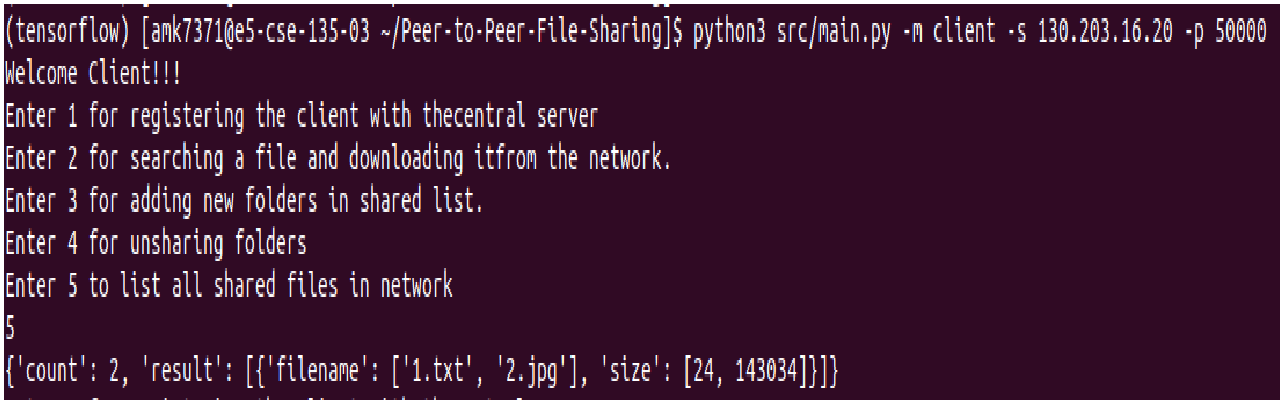
At peer -



```
(tensorflow) [amk7371@e5-cse-135-02 ~/Peer-to-Peer-File-Sharing]$ python3 src/main.py -m client -s 130.203.16.20
Welcome Client!!!
Enter 1 for registering the client with thecentral server
Enter 2 for searching a file and downloading itfrom the network.
Enter 3 for adding new folders in shared list.
Enter 4 for unsharing folders
Enter 5 to list all shared files in network
1
Welcom Client for registration!!!
Please enter the directory path of which you want to share its files.
upload
Congratulations you have been registered successfully.
[*] You will now be put to the listening state.
[*] Started listening on 130.203.16.21 : 45000
```

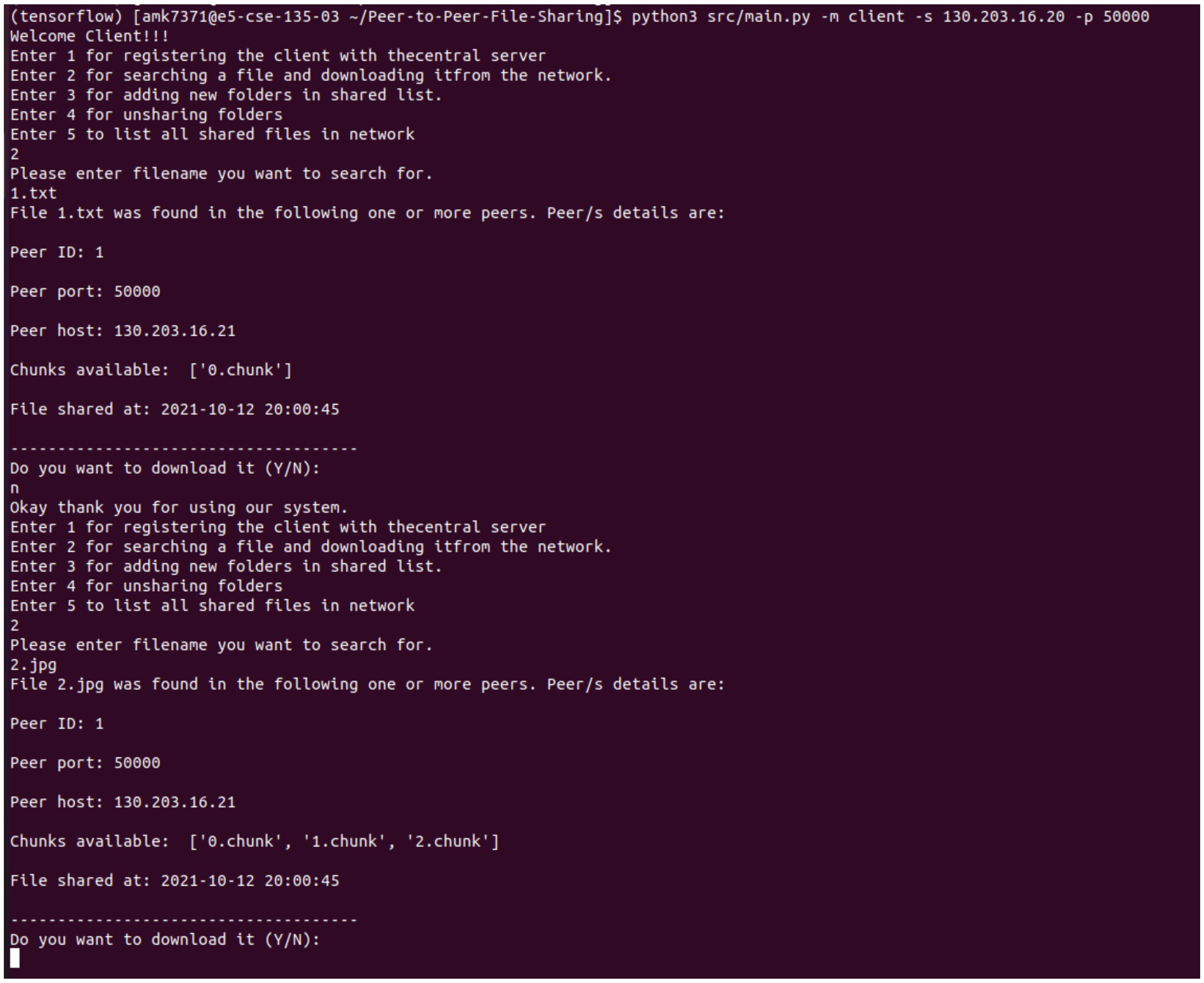
2. File List request: Peer send a command to central server. And in response: it recieves a count of total file and list of all files shared in the network along with its size.

At peer 2 (e5-135-03) -



```
(tensorflow) [amk7371@e5-cse-135-03 ~/Peer-to-Peer-File-Sharing]$ python3 src/main.py -m client -s 130.203.16.20 -p 50000
Welcome Client!!!
Enter 1 for registering the client with thecentral server
Enter 2 for searching a file and downloading itfrom the network.
Enter 3 for adding new folders in shared list.
Enter 4 for unsharing folders
Enter 5 to list all shared files in network
5
{'count': 2, 'result': [{'filename': ['1.txt', '2.jpg'], 'size': [24, 143034]}]}
```

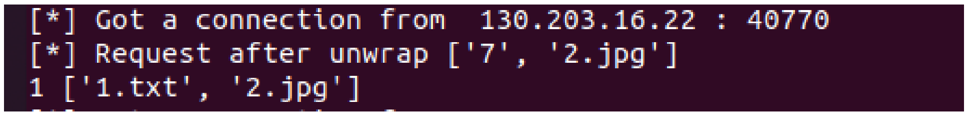
3. File Locations Request: When the peer send the file location request to central server, in return the server respond with the location of each chunk in every machine along with its total size and timestamp of chunk sharing.



```
(tensorflow) [amk7371@e5-cse-135-03 ~/Peer-to-Peer-File-Sharing]$ python3 src/main.py -m client -s 130.203.16.20 -p 50000
Welcome Client!!!
Enter 1 for registering the client with thecentral server
Enter 2 for searching a file and downloading itfrom the network.
Enter 3 for adding new folders in shared list.
Enter 4 for unsharing folders
Enter 5 to list all shared files in network
2
Please enter filename you want to search for.
1.txt
File 1.txt was found in the following one or more peers. Peer/s details are:

Peer ID: 1

Peer port: 50000

Peer host: 130.203.16.21

Chunks available:  ['0.chunk']

File shared at: 2021-10-12 20:00:45

-------------------------------------
Do you want to download it (Y/N):
n
Okay thank you for using our system.
Enter 1 for registering the client with thecentral server
Enter 2 for searching a file and downloading itfrom the network.
Enter 3 for adding new folders in shared list.
Enter 4 for unsharing folders
Enter 5 to list all shared files in network
2
Please enter filename you want to search for.
2.jpg
File 2.jpg was found in the following one or more peers. Peer/s details are:

Peer ID: 1

Peer port: 50000

Peer host: 130.203.16.21

Chunks available:  ['0.chunk', '1.chunk', '2.chunk']

File shared at: 2021-10-12 20:00:45

-------------------------------------
Do you want to download it (Y/N):
```

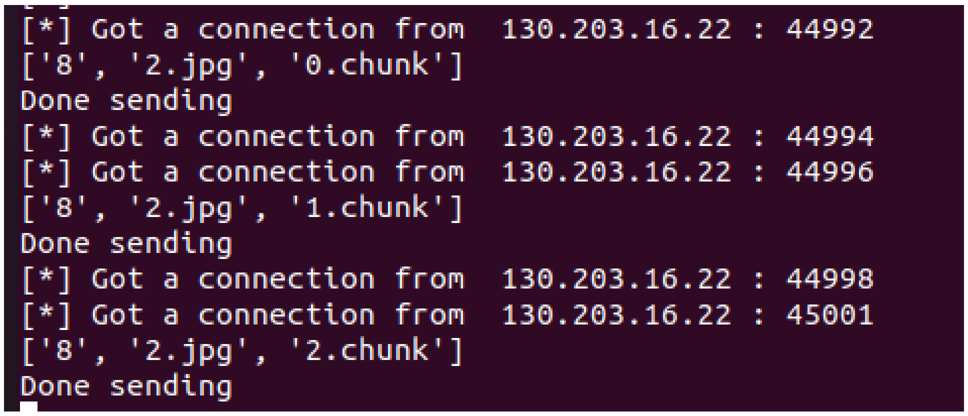
4. Download Request: In response to peer request, server share the list of addresses of each chunk with the pee. Then further requesting peer tries connecting with one of the peers in the list of each chunk and tries downloading the chunk from them.

At Central server -



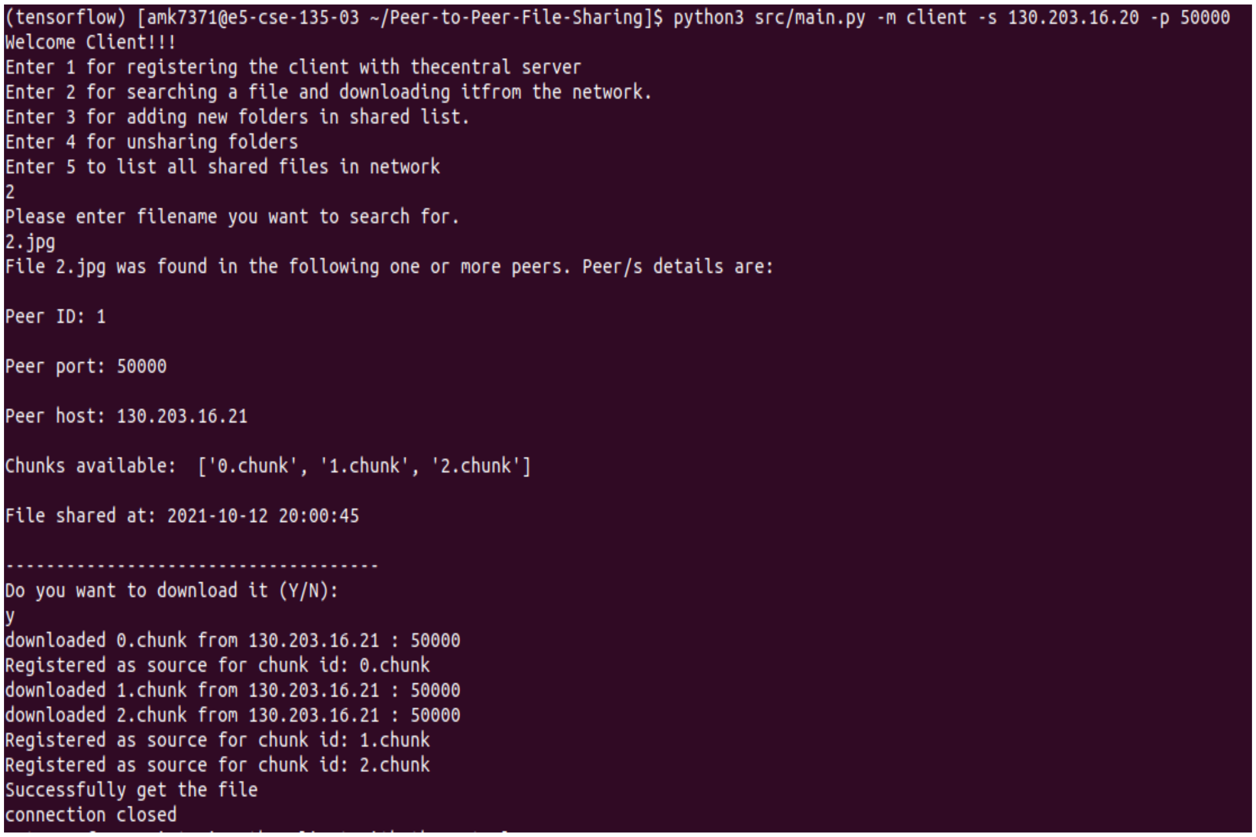
```
[*] Got a connection from  130.203.16.22 : 40770
[*] Request after unwrap ['7', '2.jpg']
1 ['1.txt', '2.jpg']
```

At peer which has the chunk :-



```
[*] Got a connection from  130.203.16.22 : 44992
['8', '2.jpg', '0.chunk']
Done sending
[*] Got a connection from  130.203.16.22 : 44994
[*] Got a connection from  130.203.16.22 : 44996
['8', '2.jpg', '1.chunk']
Done sending
[*] Got a connection from  130.203.16.22 : 44998
[*] Got a connection from  130.203.16.22 : 45001
['8', '2.jpg', '2.chunk']
Done sending
```

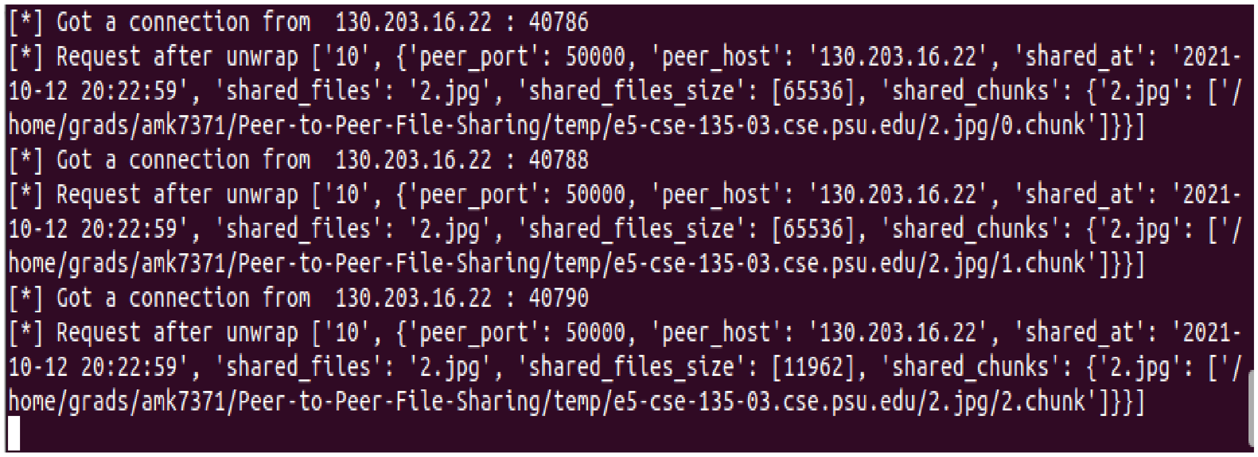
At downloading peer (e5-cse135-03)



```
(tensorflow) [amk7371@e5-cse-135-03 ~/Peer-to-Peer-File-Sharing]$ python3 src/main.py -m client -s 130.203.16.20 -p 50000
Welcome Client!!!
Enter 1 for registering the client with thecentral server
Enter 2 for searching a file and downloading itfrom the network.
Enter 3 for adding new folders in shared list.
Enter 4 for unsharing folders
Enter 5 to list all shared files in network
2
Please enter filename you want to search for.
2.jpg
File 2.jpg was found in the following one or more peers. Peer/s details are:

Peer ID: 1

Peer port: 50000

Peer host: 130.203.16.21

Chunks available:  ['0.chunk', '1.chunk', '2.chunk']

File shared at: 2021-10-12 20:00:45

-----------------------------------
Do you want to download it (Y/N):
y
downloaded 0.chunk from 130.203.16.21 : 50000
Registered as source for chunk id: 0.chunk
downloaded 1.chunk from 130.203.16.21 : 50000
downloaded 2.chunk from 130.203.16.21 : 50000
Registered as source for chunk id: 1.chunk
Registered as source for chunk id: 2.chunk
Successfully get the file
connection closed
```

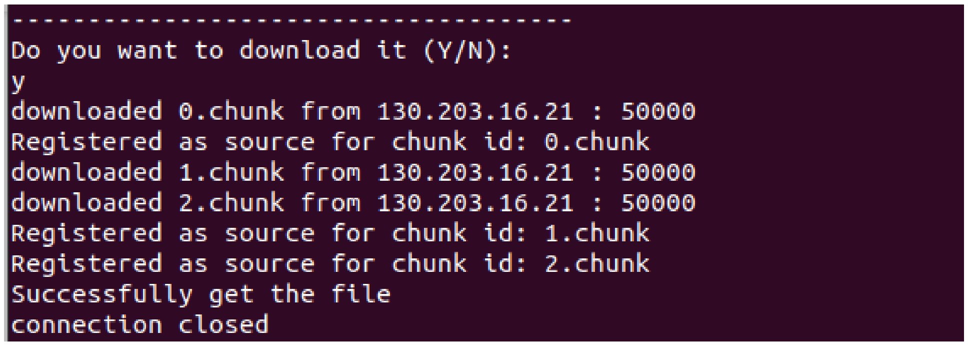
Chunk register: Once the peer downloads the chunk, it also register itself as a source for that chunk at central server by sending a command.

At central server-



```
[*] Got a connection from  130.203.16.22 : 40786
[*] Request after unwrap ['10', {'peer_port': 50000, 'peer_host': '130.203.16.22', 'shared_at': '2021-
10-12 20:22:59', 'shared_files': '2.jpg', 'shared_files_size': [65536], 'shared_chunks': {'2.jpg': ['/
home/grads/amk7371/Peer-to-Peer-File-Sharing/temp/e5-cse-135-03.cse.psu.edu/2.jpg/0.chunk']}}]
[*] Got a connection from  130.203.16.22 : 40788
[*] Request after unwrap ['10', {'peer_port': 50000, 'peer_host': '130.203.16.22', 'shared_at': '2021-
10-12 20:22:59', 'shared_files': '2.jpg', 'shared_files_size': [65536], 'shared_chunks': {'2.jpg': ['/
home/grads/amk7371/Peer-to-Peer-File-Sharing/temp/e5-cse-135-03.cse.psu.edu/2.jpg/1.chunk']}}]
[*] Got a connection from  130.203.16.22 : 40790
[*] Request after unwrap ['10', {'peer_port': 50000, 'peer_host': '130.203.16.22', 'shared_at': '2021-
10-12 20:22:59', 'shared_files': '2.jpg', 'shared_files_size': [11962], 'shared_chunks': {'2.jpg': ['/
home/grads/amk7371/Peer-to-Peer-File-Sharing/temp/e5-cse-135-03.cse.psu.edu/2.jpg/2.chunk']}}]
```

At downloading peer-



```
-------------------------------------
Do you want to download it (Y/N):
y
downloaded 0.chunk from 130.203.16.21 : 50000
Registered as source for chunk id: 0.chunk
downloaded 1.chunk from 130.203.16.21 : 50000
downloaded 2.chunk from 130.203.16.21 : 50000
Registered as source for chunk id: 1.chunk
Registered as source for chunk id: 2.chunk
Successfully get the file
connection closed
```

Code Structure

Main.py: represents the top-level program of our system, it parses all the argument and instantiates a server or a peer based on its run-time arguments.

Server.py: represents the high-level model central server. It defines the interface for communication between the peer and the server.

Controller.py: all the functioning of the central server is defined here.

Data_object.py: is the metadata stored in the server.py

Peer.py: it represents the peer, all the functionalities such as chunk download and scheduling, choking/unchoking, listening is defined here.

Helper.py: all the functions which are used inside the peer class and not necessarily needed to be the member functions are defined here.

Constants.py: defines all the constants in the system.

Code walk-

The following code snippet creates a server or a peer based on user input.

```
#Build node for the P2P system
if arguments['mode'].lower() == 'server':
  print("Port: ",port)
  build_server(port)
elif arguments['mode'].lower() == 'client':
  if arguments['server_ip'] is None:
     print("Error!!! Please set \"server-ip\" to valid value")
     exit(1)
  ip = arguments['server_ip']
  build_client(ip, port)
else:
  print ("Error!! Please use server/client as value for mode")
  exit(1)
```

If user demands the build server, it creates an object of server classes which book-keeps all the file metadata and run it as daemon thread (always listening) using following code snippet.

```
def build_server(port):
    print ("Welcome Server!!!")
    hostname = socket.gethostname()
    ip_address = socket.gethostbyname(hostname)
    print("Host name: ",hostname,ip_address,port)
    server = Server(socket, port, ip_address)
    server.run()
```

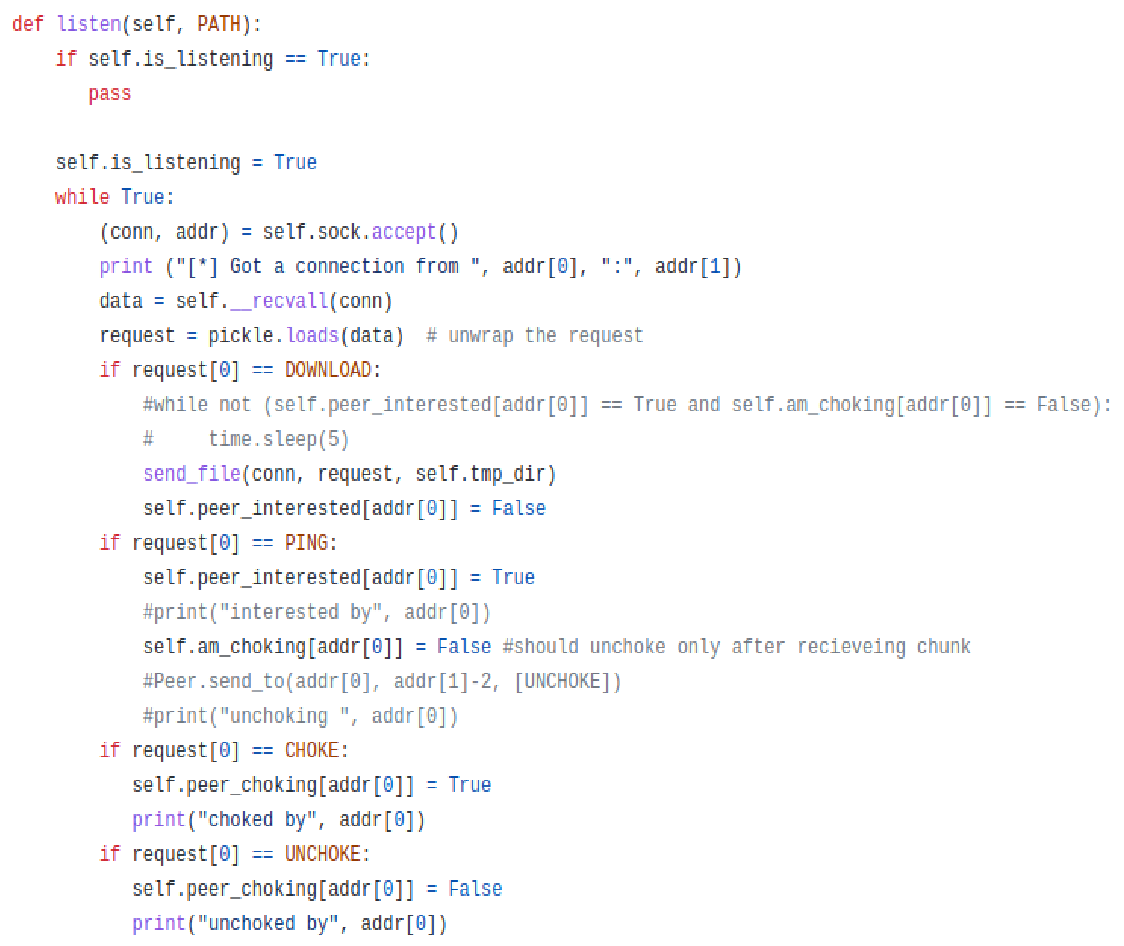
According to the following snippet, If user demands to register the node as peer, it asks for the dir location where all the files exist and create a dictionary with network config, filename etc. And split the file into multiple chunks based on the size given in constant.py

```
try:
    shared_files = [f for f in listdir(PATH) if os.path.isfile(path.join(PATH, f))]
    shared_files_size = [os.path.getsize(path.join(PATH,f)) for f in listdir(PATH) if os.path.isfile(path.join(PATH, f))]
    if len(shared_files) != 0:
       shared_chunks = peer.preprocess_reg_file(PATH) #divide into chunks
       REGISTERED_SUCCESSFULLY = peer.data_object.register
       sharing_datetime = datetime.datetime.fromtimestamp(int(time.time())).strftime('%Y-%m-%d %H:%M:%S')
       data_object = dict(peer_port=server_port, peer_host=client_ip, shared_files=shared_files, shared_files_size=shared_files_size, shared_at=sharing_dat
       if REGISTERED_SUCCESSFULLY:
            DATA_INSERTED = peer.data_object.append_data(data_object)
            if DATA_INSERTED:
               print ("Congratulations you have been registered successfully.\n" \
                      "[*] You will now be put to the listening state.\n" \
                      "[*] Started listening on", client_ip, ":", server_port)
               i_thread = threading.Thread(target=peer.listen, args=(PATH,))
               i_thread.start() #tpeer.listen(PATH)  # block until you receive request
            else:
                print ("There was an error while inserting your data.")
    else:
       print ("You cannot share empty directory.")
except Exception as exc:
    print("Caught exception: %s" %str(exc))
```

In the given snippet, server is running a forever thread,, where it keep listening to the socket, and once it receives any meaningful message, it takes an action according to the command name.

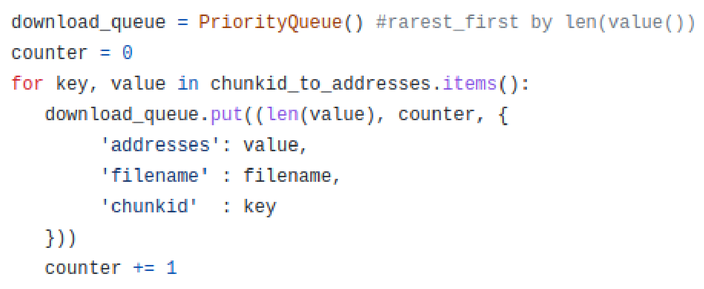
```
def run(self):
    while True:
        (conn, addr) = self.sock.accept()  # accept incoming call
        print ("[*] Got a connection from ", addr[0], ":", addr[1])
        data = self.__recvall(conn)  # fetch data from peer
        request = pickle.loads(data)  # unwrap the request
        print ("[*] Request after unwrap", request)
        if request[0] == REGISTER:  # create a list
            self.semaphore.acquire()
            register(conn, self.setOfLists)
            self.semaphore.release()

        elif request[0] == APPEND:  # append request
            self.semaphore.acquire()
            append(conn, request, self.setOfLists)
            self.semaphore.release()

        elif request[0] == GETVALUE:  # read request
            list_id = request[1]  # fetch list_id
            result = self.setOfLists[list_id]  # get the elements
            conn.send(pickle.dumps(result))  # return the list
        # -
        elif request[0] == STOP:  # request to stop
            print (self.setOfLists)
            conn.close()  # close the connection  #-
            break  # -
        elif request[0] == SEARCH:
            self.semaphore.acquire()
            found_boolean, file_object = search(request[1], self.setOfLists)
            if found_boolean:
                conn.send(pickle.dumps([found_boolean, file_object]))
            else:
                conn.send(pickle.dumps([found_boolean]))
            self.semaphore.release()
        elif request[0] == ENUMERATE:
           result_file_list = file_list(self.setOfLists)
           conn.send(pickle.dumps(result_file_list))
        elif request[0] == REGISTER_CHUNK:
           self.semaphore.acquire()
           register_chunk(conn, request, self.setOfLists)
           self.semaphore.release()
```

When a node is registered as a peer, it launches a thread which keep listening to the socket for any request from other nodes. The above-mentioned functionality is mentioned in the following code snippet.



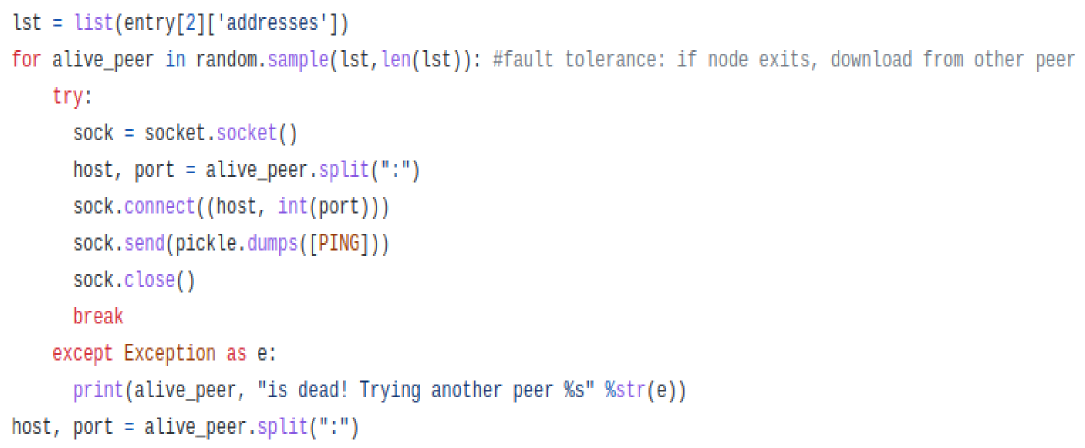
```python
def listen(self, PATH):
    if self.is_listening == True:
       pass

    self.is_listening = True
    while True:
        (conn, addr) = self.sock.accept()
        print ("[*] Got a connection from ", addr[0], ":", addr[1])
        data = self.__recvall(conn)
        request = pickle.loads(data)  # unwrap the request
        if request[0] == DOWNLOAD:
            #while not (self.peer_interested[addr[0]] == True and self.am_choking[addr[0]] == False):
            #     time.sleep(5)
            send_file(conn, request, self.tmp_dir)
            self.peer_interested[addr[0]] = False
        if request[0] == PING:
            self.peer_interested[addr[0]] = True
            #print("interested by", addr[0])
            self.am_choking[addr[0]] = False #should unchoke only after recieveing chunk
            #Peer.send_to(addr[0], addr[1]-2, [UNCHOKE])
            #print("unchoking ", addr[0])
        if request[0] == CHOKE:
           self.peer_choking[addr[0]] = True
           print("choked by", addr[0])
        if request[0] == UNCHOKE:
           self.peer_choking[addr[0]] = False
           print("unchoked by", addr[0])
```

We created a priority_queue to create download_taskqueue with rarest chunk first as the scheme in the below code snippet.



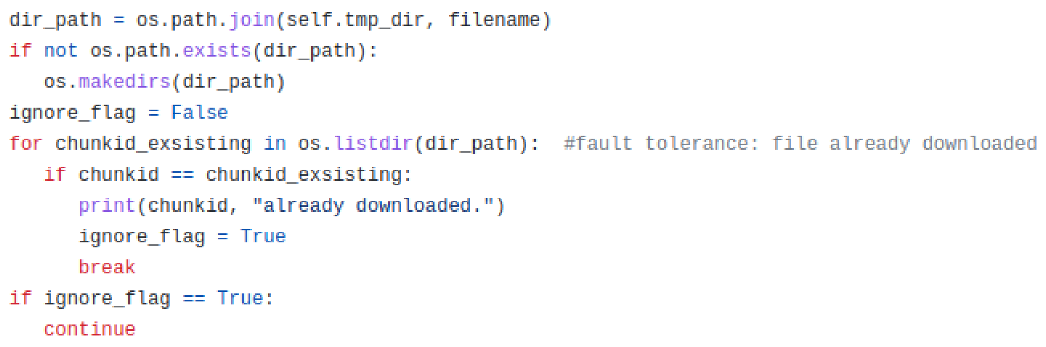
```python
download_queue = PriorityQueue() #rarest_first by len(value())
counter = 0
for key, value in chunkid_to_addresses.items():
   download_queue.put((len(value), counter, {
        'addresses': value,
        'filename' : filename,
        'chunkid'  : key
   }))
   counter += 1
```

The following code snippet randomly pick an address for the chunk and try pinging to make the system fault tolerant (if source peer died in between). If the PING message fails, then source node had already crashed, and it tries a new address.



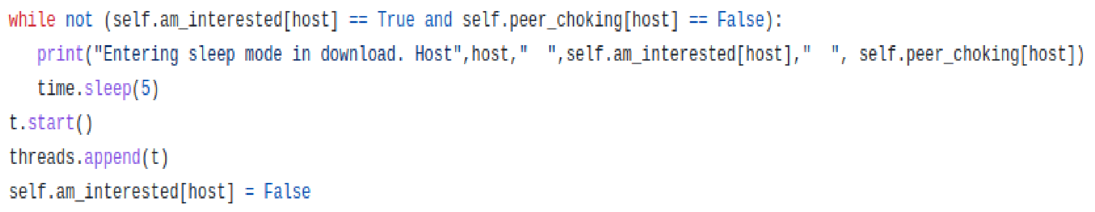
```
lst = list(entry[2]['addresses'])
for alive_peer in random.sample(lst,len(lst)): #fault tolerance: if node exits, download from other peer
  try:
   sock = socket.socket()
   host, port = alive_peer.split(":")
   sock.connect((host, int(port)))
   sock.send(pickle.dumps([PING]))
   sock.close()
   break
  except Exception as e:
   print(alive_peer, "is dead! Trying another peer %s" %str(e))
host, port = alive_peer.split(":")
```

If the destination node crashes in between and reconnects, then the following code snippet access the temp directory to figure out the remaining chunks and only download the remaining ones. And thus making system fault tolerant!



```
dir_path = os.path.join(self.tmp_dir, filename)
if not os.path.exists(dir_path):
   os.makedirs(dir_path)
ignore_flag = False
for chunkid_exsisting in os.listdir(dir_path):  #fault tolerance: file already downloaded
   if chunkid == chunkid_exsisting:
      print(chunkid, "already downloaded.")
      ignore_flag = True
      break
if ignore_flag == True:
   continue
```

For choking/unchoking :If the peer is interested in the connection with the other peer and the other peer is not choking the node then destination peer downloads the chunk or else it sleeps for few seconds until the condition became false.



```
while not (self.am_interested[host] == True and self.peer_choking[host] == False):
   print("Entering sleep mode in download. Host",host,"  ",self.am_interested[host],"  ", self.peer_choking[host])
   time.sleep(5)
t.start()
threads.append(t)
self.am_interested[host] = False
```

Similarly, in the upload path if I'm not choking the peer and the other node is also interested in the connection with me, I send the chunk of the file.

```
if request[0] == DOWNLOAD:
    while not (self.peer_interested[addr[0]] == True and self.am_choking[addr[0]] == False):
        #print(" entering sleep mode in download. ",addr[0])
        time.sleep(5)
    send_file(conn, request, self.tmp_dir)
    self.peer_interested[addr[0]] = False
```