MacTella's Flow-Chart

We first define a Class named Peer, which has IP_address and Port attributes

```
class Peer(object):
    IP_address = None
    Port = None
```

We then define a Class named Files, which has 4 attributes, "name", "Path", "size", and "type"

```
class Files(object):
    name = None
    full_path = None
    size = None
    type = None
```

Functions

Define a funtion "getList" to loop through all the file in the current dictory, for each file we find, a new "Files" object is created and added to "List_of_files" for storing the information needed for a transfer and the "LISTL" and "LISTR" functions. Using the OS library, we can get the current directory and sizes of files with the full file path as a parameter.

```
def getList():
    List_of_files = []
    curruent_path = os.getcwd()
    file_found = False
    response = ''
    for files in os.listdir(curruent_path):
        new_file = Files()
        new_file.name = files
        new_file.full_path = curruent_path + "\\" + files
        new_file.size = os.path.getsize(curruent_path + "\\" + files)
        fileName, fileExtension = os.path.splitext(new_file.full_path)
        if (fileExtension == '.mov'):
            new_file.type = "viedo"
            List_of_files.append(new_file)
        elif (fileExtension == '.mp3'):
            new_file.type = "music"
            List_of_files.append(new_file)
        elif (fileExtension == '.jpg'):
            new_file.type = "picture"
            List_of_files.append(new_file)
    return List_of_files
```

Define a function to send data using new port to any peer. First define a new socket using socket.socket(socket.AF_INET, socket.SOCK_STREAM), then try to "sock.connect" to the server address provided and use "sendall" to send the message.

```
def send_using_new_socket(IP,port,message):
    server_address = (IP,int(port))

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        sock.connect(server_address)
        sock.sendall(message)
    except:
        pass
```


Jingnan Chen 1073196, Wei Zhang 0951321

MacTella's Flow-Chart

Global Variable

Define a global variable "LIST_of_Peers" to store all the peers known to this peer, and global variables to store its HOST IP address and PORT number.

```
List_of_Peers = []
HOST, PORT = "localhost", 10000
```

"CLIENT-SIDE"

Entering Main, a sever is defined ("server = ThreadedTCPServer((HOST, PORT), ThreadedTCPRequestHandler)") and started ("server_thread.start()") to listen to responses from other peers.

Then a "Sock" object and "server_address" are defined for connecting to other peers later. One known peer is hardcoded to begin with.

```
if __name__ == "__main__":
    quit_server = False
    # Port 0 means to select an arbitrary unused port

    print "\nStart Threaded-Server on PORT %s " % PORT

    server = ThreadedTCPServer((HOST, PORT), ThreadedTCPRequestHandler)
    ip, port = server.server_address

    # Start a thread with the server -- that thread will then start one
    # more thread for each request
    server_thread = threading.Thread(target=server.serve_forever)

    # Terminate the server when the main thread terminates
    # by setting daemon to True
    server_thread.daemon = True
    server_thread.start()
    print "Main Server using thread %s " % server_thread.name


    # Create a TCP/IP socket
    sock = None
    server_address = ''

    # Add 1 known peer to begin with
    known_peer = Peer()
    known_peer.IP_address = 'localhost'
    known_peer.Port = 10001
    List_of_Peers.append(known_peer)


    # Get the list of files in the current directory for later use
    List_of_files = getList()
```

QUIT Command

Jingnan Chen 1073196, Wei Zhang 0951321

MacTella's Flow-Chart

A while True loop is used to continuously collect user input using "raw_input("Enter your Command: \n")"

If "QUIT" is entered, first check if there is any threads (connections from other peer) exists, if so wait for them to finish, the user can type "FORCEQUIT" to forcefully close, or wait for any threads to finish and quit the program.

```
while True:
    # using while loop and raw_input to allow admin to quit the server
    command = raw_input("Enter your Command: \n")
    if(command == "QUIT"):
        if(thread_counter ==0):
            # if no connections the server closes by server.shutdown and quit()
            print 'Main server thread shutting down the server and terminating'
            server.shutdown()
            quit()
        else:
            print 'Waiting for threads to finish... \n'
            while(thread_counter !=0):
                # if there are connection, the admin can still force quit the server
                force_comment = raw_input("Type FORCEQUIT to quit abruptly. \n")
                if(force_comment == "FORCEQUIT"):
                    print 'Bye'
                    os._exit(0)
                quit_server = True
            quit()
```

LISTL Command

If "LISTL" is entered, call the getList() function discussed above to get a list of files in the current directory, and use for-loop to go through each Files object and print its attributes to the screen. If there is no objects in the list, print "There is no files in the directory" instead.

```
elif(command == "LISTL"):
    #get local files
    feedback = ""
    List_of_files=getList()
    for files in List_of_files:
        # prepare the response
        feedback = feedback + str(files.type)+ "\t " + str(files.name) + "\t size: " + str(files.size) + "
    if(feedback == ""):
        # send the response using request.sendall
        print "There is no files in the directory"
    else:
        print feedback
```

LISTR Command

If "LISTR" is entered, define the sock object declared earlier, use "sock.connect(address)" to connect to the server address provided and print to the screen what is received from the peer using "sock.recv(1024)"

```
elif(command.startswith("LISTR")):
     if(len(command.split(',')) == 3):
          #inputs are okay
          server_address = (command.split(',')[1],int((command.split(',')[2])))
          sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
          try:
               sock.connect(server_address)
               sock.sendall("LISTR")

               data = sock.recv(1024)
               if data:
                    print data
          except:
               print "Could not connect to the IP and Port provided"
```

Jingnan Chen 1073196, Wei Zhang 0951321

MacTella's Flow-Chart

SEARCH Command

If "SEARCH" is entered, define the sock object declared earlier, use "sock.connect(address)" to connect to the server address provided. This peer's **IP and port number is appended** to the end of the command to send to the peers in the network, so that they will know where to send back the message to. Main logic is in the server-sided code, disscussed later on in the flow chart.

```
elif(command.startswith("SEARCH")):
    if(len(command.split(',')) == 3):
        #inputs are okay
        for peer in List_of_Peers:
            try:
                server_address = (peer.IP_address,peer.Port)
                sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                sock.connect(server_address)
                sock.sendall(command + "," + HOST + "," + str(PORT))
                sock.close()

            except:
                print "Could not connect to " + peer.IP_address + "."

    else:
        print "please provide the file name and time to live hop number seperated by comma. \n"
```

DISCOVER Command

If "DISCOVER" is entered, define the sock object declared earlier, use "sock.connect(address)" to connect to the server address provided. This peer's **IP and port number is appended** to the end of the command to send to the peers in the network, so that they will know where to send back the message to. Main logic is in the server-sided code, disscussed later on in the flow chart.

```
elif(command.startswith("DISCOVER")):
    if(len(command.split(',')) == 2):
        for peer in List_of_Peers:
            try:
                server_address = (peer.IP_address,peer.Port)
                sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                sock.connect(server_address)
                sock.sendall(command + "," + HOST + "," + str(PORT))
                sock.close()
            except:
                print "Could not connect to " + str(peer.IP_address)
    else:
        print "please provide the time to live hop number seperated by comma. \n"
```

GET Command

Jingnan Chen 1073196, Wei Zhang 0951321

MacTella's Flow-Chart

If "GET" is entered, define the sock object declared earlier, use "sock.connect(address)" to connect to the server address provided. If the connection is successful, then set connected Boolean to True, else set to False

```
elif(command.startswith("GET")):
    if(len(command.split(',')) == 4):
        #inputs are okay
        connected = False
        server_address = (command.split(',')[2],int((command.split(',')[3])))
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            sock.connect(server_address)
            connected = True
        except:
            print "Could not connect to the server address provided."
            connected = False
```

If "connected" is True, then received the data from the peer once, to check if there is an error from the peer's side, if no error, then the received data is the file size(we defined at the server-side), open a file in write-byte mode "f1 = open(command.split(',')[1],'wb')".

```
if(connected):
    sock.sendall(command)
    data = sock.recv(64)
    if(data.startswith("ERROR:")):
        print >>sys.stderr, '"%s"' %data
    else:
        filesize = int(data)
        amount_received = 0
        f1 = open(command.split(',')[1],'wb')
```

While the amount received is less than the file size, keep on receiving using "data = sock.recv(128)", then write to the file using "f1.write(data)", and increment the amount received variable. If connection break during transmission, close the file and delete it using "f1.close()", and "os.remove(f1.name)".

```
while amount_received < filesize:
    try:
        data = sock.recv(128)
        f1.write(data)
        amount_received = len(data) + amount_received
    except:
        f1.close()
        os.remove(f1.name)
        sock = None
        break
```

Finally, after the while loop, check if amount_received is equal to file size, if True then close the file. And print successful message to user.

```
if(amount_received == filesize):
    print "File Received"
    f1.close()
```


Jingnan Chen 1073196, Wei Zhang 0951321

MacTella's Flow-Chart

RESET Command

If "RESET" is entered, clear the list of peers "List_of_Peers = []" and hardcode the know peer back into the list.

```
elif(command.startswith("RESET")):
    List_of_Peers = []
    known_peer = Peer()
    known_peer.IP_address = 'localhost'
    known_peer.Port = 10001
    List_of_Peers.append(known_peer)
    print "List of Peers has been reseted"
#command not recognized
else:
    print "command entered is not recognized"
```

If the entered command is none of the above, then print "command entered is not recognized" back to the user, shown above.

SERVER-SIDE

Upon server receiving any data, "class ThreadedTCPRequestHandler(SocketServer.BaseRequestHandler):" will be called to handle the data.

First increase the thread_counter for the "QUIT" command and receive the data then strip off any spaces using "self.request.recv(1024)"

```
class ThreadedTCPRequestHandler(SocketServer.BaseRequestHandler):

    def handle(self):
        global thread_counter
        thread_counter  += 1

        cur_thread = threading.current_thread()  # identify current thread
        thread_name = cur_thread.name   # get thread-number in python

        data = self.request.recv(1024)
        data = data.strip()
```

LISTR Received

Jingnan Chen 1073196, Wei Zhang 0951321

MacTella's Flow-Chart

If data is not empty, then if data starts with "LISTR". Call the "getList()" function for a list of files in the directory and for each object in the list send the file's attributes using "self.request.sendall(data)" else send no files in the directory.

```
if(data != ''):
    if(data.startswith('LISTR')):
        response = ""
        # get list of files using the fucntion define above
        List_of_files=getList()
        for files in List_of_files:
            # prepare the response
            response = response + str(files.type)+ "\t " + str(files.name) + "\t size: " + str(files.size)
        if(response == ""):
            # send the response using request.sendall
            self.request.sendall("There is no files in the directory")
        else:
            self.request.sendall(response)
            response = ""
```

DSICOVER Received

If data starts with "DISCOVER". Recover the IP address and port number of the peer requesting "DSICOVER", decrement the time to live variable, and send back to the requesting peer an alive message "DISCOV_Result". If time to live is zero, then drop the request.

```
elif(data.startswith("DISCOVER")):
    #print 'received discover'
    TTL = int(data.split(',')[1])
    main_IP = data.split(',')[2]
    main_port = int((data.split(',')[3]))

    if(TTL > 0):
        TTL -= 1
        response = "DISCOV_Result," + HOST + "," + str(PORT)
        data = data.split(',')[0] + ","  + str(TTL) + "," + main_IP + "," + str(main_port)
        send_using_new_socket(main_IP, main_port, response)
```

Then forward the message with decremented time to live to all its peers **except for the requesting peer**.

```
for peer in List_of_Peers:
    if(peer.IP_address != main_IP or peer.Port != main_port):
        send_using_new_socket(peer.IP_address,peer.Port, data)
```

Alive Message "DISCO_Result" Received

Jingnan Chen 1073196, Wei Zhang 0951321

MacTella's Flow-Chart

If data starts with "DISCOV_Result", an "alive" message. Check if the peer is already in the list. If not create a new peer object and append it to the list.

```
elif(data.startswith("DISCOV_Result")):
    #print "disocover result"
    new_peer = True
    new_IP_address = data.split(',')[1]
    new_port_number = int(data.split(',')[2])

    for peer in List_of_Peers:
        if(peer.IP_address == new_IP_address and peer.Port == new_port_number):
            new_peer = False


    if(new_peer == True):
        a_new_peer = Peer()
        a_new_peer.IP_address = new_IP_address
        a_new_peer.Port = new_port_number
        List_of_Peers.append(a_new_peer)
        print "IP:" + str(a_new_peer.IP_address) +  ",Port:" + str(a_new_peer.Port) + " has been added
    else:
        print "IP:" + str(new_IP_address) +  ",Port:" + str(new_port_number) + " is already in the lis
```

SEARCH Received

If data starts with "SEARCH", Recover the filename and the IP address and port number of the peer requesting "DSICOVER", then get the list of files in the current directory.

```
elif(data.startswith("SEARCH")):
    #print 'received search'
    TTL = int(data.split(',')[2])
    filename = data.split(',')[1]
    main_IP = data.split(',')[3]
    main_port = data.split(',')[4]
    List_of_files = getList()
```

If Time to live is not zero, decrement the variable and loop through the list of files to search for the filename. If match is found send to the requesting peer a found message "SEAR_Result" with its IP address and Port number.

```
if(TTL > 0):
    TTL = TTL - 1
    data = data.split(',')[0] + "," + filename + "," + str(TTL) + "," + main_IP + "," + main_port

    for files in List_of_files:
        if files.name == filename:
            #file found
            response =  "SEAR_Result" + "," +  HOST + "," + str(PORT)
            send_using_new_socket(main_IP,main_port, response)
            break
```

Then forward the message to all its peers except the requesting peer, to check If any another peers has the same file.

```
for peer in List_of_Peers:
    if(peer.IP_address != main_IP or peer.Port != main_port):
        send_using_new_socket(peer.IP_address,peer.Port, data)
```

MacTella's Flow-Chart

"SEAR_Result" (file found) Received

If data starts with "SEAR_Result", a "found" message. Print the IP address and port number contained in the message.

```
elif(data.startswith("SEAR_Result")):
    #print "sear_result"
    #print data
    print "File has been found at IP_Address: " + data.split(',')[1] +" Port: " + data.split(',')[2] + "\n"
```

GET Received

If data starts with "GET", get the list of files in the current directory, and check if there are enough parameters provided. Then, loop through the list of files to find a matching file name.

```
elif(data.startswith("GET")):
    #get file list
    List_of_files=getList()
    if(len(data.split(',')) == 4):
        filename = data.split(',')[1]
        file_found = False
        #loop through the list to check if the file exist
        for files in List_of_files:
            if(files.name == filename):
```

If a match is found, first send the file size using "self.request.sendall(str(files.size))", then open the file in read-byte mode, and for each line in the file, send that line using "self.request.sendall(line)". Close the files after the transfer is done.

```
if(files.name == filename):
    file_found = True
    #send the file size to the client for error handling
    self.request.sendall(str(files.size))

    #change the directoy just in case
    os.chdir(files.full_path[:len(files.full_path) - len(files.name)])

    print "\nREAD Command From %s" %thread_name
    print "Sending " + str(filename) + " to %s" %thread_name

    # open the file in read byte mode for transfer
    f1 = open(files.name, 'rb')

    for line in f1:
        #send each line of the file using a for loop
        self.request.sendall(line)

    # close the file after transfer
    f1.close()
```

If file name was not found, send back "ERROR:" message to the requesting peer using "sendall"

```
if(file_found == False):
    #if file is not found, send error to the client for handling
    response = "ERROR: could not find the file in the server"
    self.request.sendall(response)
```


Jingnan Chen 1073196, Wei Zhang 0951321

MacTella's Flow-Chart