IP PROJECT 1:
Ashlesha Atrey 200203500
Sai Kiran 200202554

Introduction:
We implemented Peer-to-Peer with Distributed Index (P2P-DI) System for Downloading RFCs. Following is the basic blocks of the project explaining the message format and the protocol.

RS Server:
First the server IP address and port are fixed and well known to all the port.
TCP_IP = '127.0.0.1'
TCP_PORT = 55567
The RS server should always be in the listening state. Whenever a new client connects to the server a new thread is opened for the server. Server can receive four queries from the client which are Register, Leave, PQuery and KeepAlive. Following is the algorithm followed for each request.

Register:
First the server looks in its database, if the client is found registered then the same cookie is given to client with the message that client is already registered with that cookie.
At that time if the flag of the client is inactive i.e. false then the flag is made active and keep alive is updated to 7200 sec. Registration number is increased by 1 and updated in database.

If the client is not found in the database, then the client is registered, and a fresh cookie is given to the client and the database is created for that client.
When the client is registered for the first time, server updates its database with the following seven parameters.
[cookie, flag=True, TTL=7200, servport, rgno, time]

Register response message:
If registered for the first time:
Version 1.1 /n Status code: Passed \n Client registered successfully with cookie:<Cookie number>
If registered again:
Version 1.1 /n Status code: Passed \n Client already registered with cookie:<cookie number which was allocated before>

Leave:
When leave is sent by the client, the client is marked as inactive by the server and the database is updated.

Version 1.1 \n Status code: Passed \n Client is marked as inactive

PQuery:
Firstly, the client keep alive value is updated to 7200sec. Database is parsed to check all the active clients and a list of active clients containing ip, serverport and the keepalive value is sent to client.

"Version 1.1 \n Status code: Passed \n Active list directory"

KeepAlive:
The client database is updated to 7200 sec. If the client was inactive then the client is made active again.

Version 1.1 \n Status code: Passed \n Client TTL set to 7200 sec

In the background whenever anyone connects to the server, let it be any client first the database is parsed and all the peers which are registered have their keepalive value compared. Before that keepalive is reduced for the amount of time that the last request from any client was received. This give us the difference in time between the last request and the current request.
Keepalive value for all the registered client is decreased by this amount.
When the keepalive value reaches <0 the database is updated, and flag is set inactive.

Client-Server:

In client-server, there is a thread open for menu function which is the main thread in the program. All other threads are daemon threads. Menu has the following options and only one option can be executed at once.
Each client is bind to a fixed IP and port which the other clients are unware of. Below is the detailed description of the menu.

Menu
0.Register to Server
1.List all the available RFC's
2.Server lookup PQUERY
3.LEAVE
4.KEEPALIVE
5.Request for RFC
6 Get active directory
7.Exit

Register to Server: Option 0

The peer opens a TCP connection to send this registration message to the RS and provide information about the port to which its RFC server listens. A new socket is opened for this application and is bind to client IP and port. Socket is connected to the RS server. Client sends a message “Register” along with the client’s server port which is in continuous listening state. This specific message format allows the server to recognize the query. The client receives the appropriate response to the register request and closes the socket.

Request message format:

Register <Server port of client>

Example response for the above query:

Version 1.1

Status code: Passed

Client registered successfully with cookie: 46

Server lookup PQUERY: Option 2

When a peer wishes to download a query, it first sends this query message to the RS (by opening a new TCP connection), and in response it receives a list of active peers that includes the hostname and RFC server port information. A new socket is opened for this application and is bind to clients IP and port. Client sends a message “PQuery”. The server response is received along with the list of all active clients.

Request message format:

PQuery

Example: Response received from the server for the above request. The fourth field of the list is a empty list which will be updated when we query for all available RFCs in the local database for that client.

Version 1.1

Status code: Passed

Active list directory

[['127.0.0.7', '9007', '7137', []], ['127.0.0.6', '9006', '7114', []], ['127.0.0.5', '9005', '7095', []], ['127.0.0.4', '9004', '7080', []], ['127.0.0.3', '9003', '7200', []], ['127.0.0.2', '9002', '7167', []]]

Leave: Option 3

When the peer decides to leave the system (i.e., become inactive), it opens a TCP connection to send this message to the RS. A new socket is opened for this application and is bind to clients IP and port. Client sends a message "Leave".

Request message format:

Leave

Example response for the above request:

Version 1.1

Status code: Passed

Client is marked as inactive

KEEPALIVE: Option 4

A peer periodically sends this message to the RS to let it know that it continues to be active; upon receipt of this message, the RS resets the TTL value for this peer to 7200. A new socket is opened for this application and is bind to clients IP and port. Client sends a message "KEEPALIVE".

Request message format:

KeepAlive

Example response for the above request:

Version 1.1

Status code: Passed

Client TTL set to 7200 sec

List all the available RFC's: Option 1

A user input is taken for the clients IP address and server port of that client. A new socket is opened, bind with the clients IP and port.

The local database of the RFC is stored in a file called content. Initially all the clients should manually update this file. Whenever a new RFC is fetched this file is updated. When the server receives this query, the request is understood by the server and it opens it local database file to fetch the list of RFC stored in its local database.

Request message:

RFCQuery P2P-DI/1.0

Host: 127.0.0.3

OS:Linux

Example response for the above request:

Version 1.1

Status code: Passed

Connection established successfully

['3405', '3492', '3501', '3530', '3550', '3711', '3720', '3730', '3783', '3801', '7234', '7235', '7301', '7348', '7469', '7540', '7541', '7567', '7725']

Get active directory: Option 6

This function fetches the list which we got from the PQuery request. When option 6 is pressed after PQuery and option 1 Request for RFC, our list is complete with 4th field updated with the list of RFCs.

This is a local RFC index request to client itself. It just prints the list.

Response to the above request:

[['127.0.0.3', '9003', '7200', []], ['127.0.0.2', '9002', '5670', ['3405', '3492', '3501', '3530', '3550', '3711', '3720', '3730', '3783', '3801', '7234', '7235', '7301', '7348', '7469', '7540', '7541', '7567', '7725']]]

Similarly, as in the RS server, the keep alive field get updated whenever any query is done by the client.

Request for RFC: Option 5

Like fetching a list, a file is fetch instead. RFC_NO is asked as an input to the user and corresponding RFC is fetched by accessing the active list directory we got from PQuery and the list of RFCs we got from GetList Query.

Request message:

GETRFC 3405 P2P-DI/1.0

Host: 127.0.0.3

OS: Linux

Response to the above request:

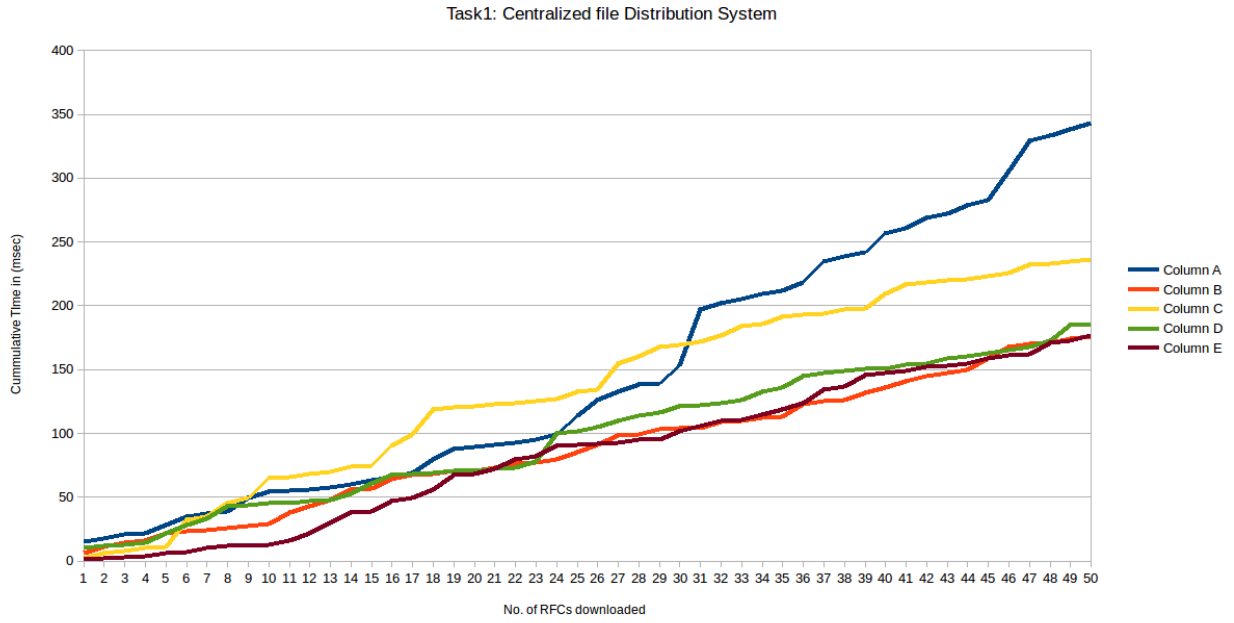
Version 1.1

Status code: Passed

File found

Task 1:

In this figure, Column A to E represents clients 1 to 5 respectively. Client 6 is our Centralized server

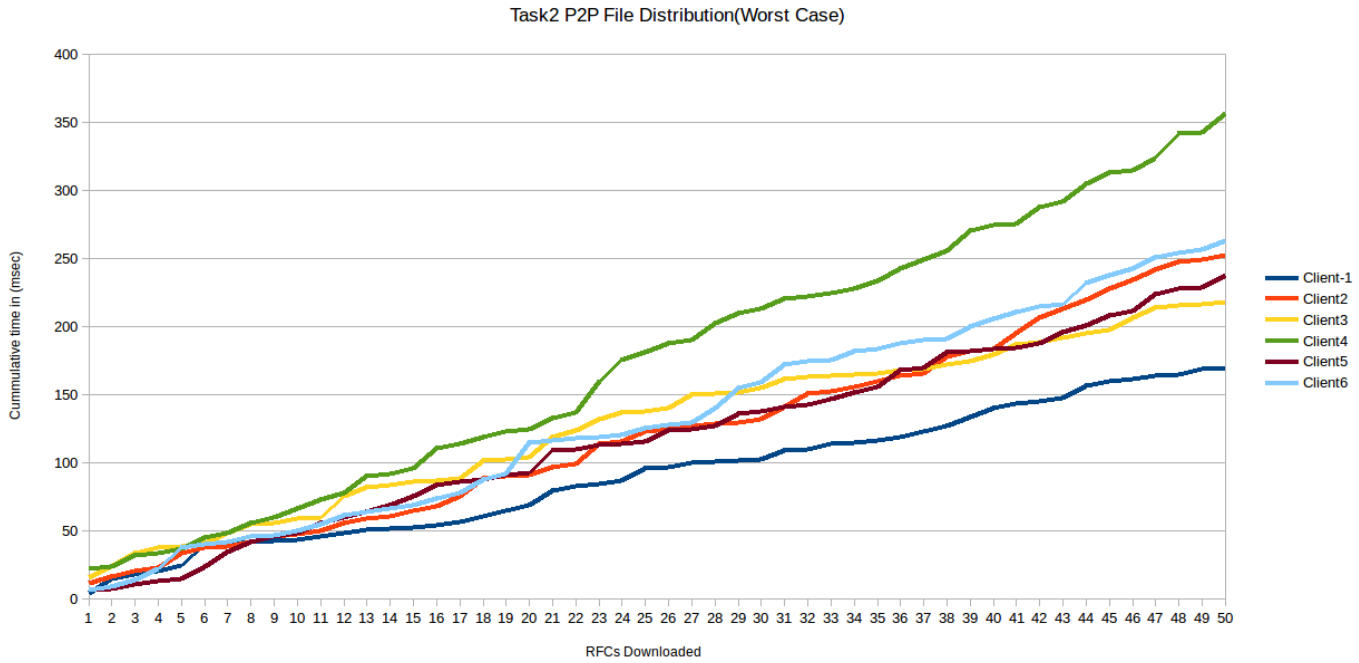


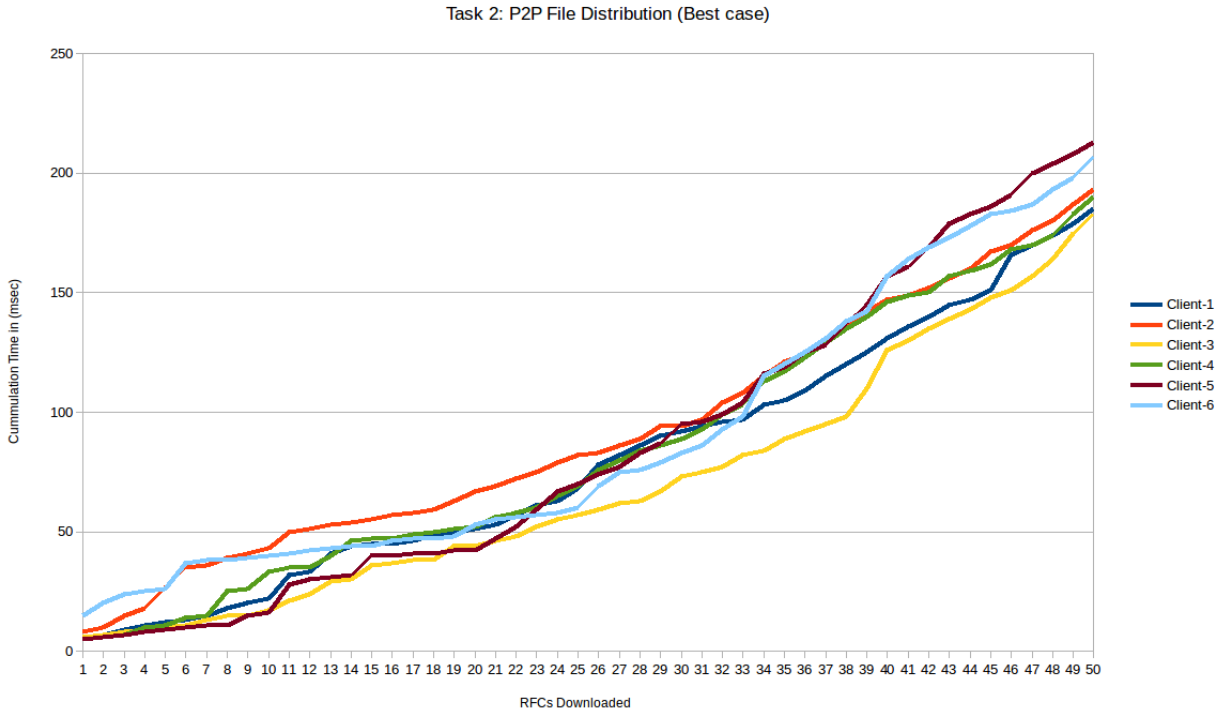
In centralized file distribution, client 6 is our centralized server. First each client should register to RS server and PQuery to the RS server.

After that client 6 have 60 RFCs with him. Each client 1-5 are asking for 50 RFC out of 60 to client 6 for at the same time. Thus client 6 will face maximum load during this time as lots of threads will running at the same time. The server must transmit one copy of the file to each of the 5 peers

Due to the process being centralized, each client will have to ask once to client 6 and hence only one socket connection will be open. This saves time of opening multiple sockets. We can see cumulative time to fetch all RFCs is around 223 milliseconds.

Task 2:


Task2 P2P File Distribution(Worst Case)
Cummulative time in (msec)
RFCs Downloaded
Client-1
Client2
Client3
Client4
Client5
Client6


Task 2: P2P File Distribution (Best case)
Cummulation Time in (msec)
RFCs Downloaded
Client-1
Client-2
Client-3
Client-4
Client-5
Client-6

In this model, each client C1-6 have 10 RFCs stored locally. Each has register to RS server and PQuery to the RS server and have the RFC index list with them.

Worst Case:

Worst case will be when each client asks every other client for list one by one in the order in which they were registered to the RS-server.

Suppose c1,c2,c3,c4,c5,c6 registered in this order. In this case C1 will ask c2 for list, C2 - C6 will ask C1 for the RFCs in the first iteration.

In second iteration, C1 will ask C3 and C2 will ask for C3 and C3-C6 will ask to C3 for list and so on.

Hence in this case they are asking the same clients for the list hence getting very few new RFCs in each iteration.

The average cumulative time to fetch RFCs is 253 milliseconds.

Best Case:

The Best case will be clients asking each other in a specific way, to get maximum RFCs in few iterations. For example, the peers registered in the fashion C1,C2,C3,C4,C5,C6. In this case, one of the best case will be to C1 asking c2 for RFC, C2 asking C3 for RFC and C4 asking C5 and C6 asking C1. Here the load is balanced among all peers and no one entity is facing a lot of load at the same time.

The next iteration C1 will ask C3 hence will also get C4 RFCs, C2 will ask C4 and so on. Hence in every iteration load is balanced and no duplicate RFCs are fetch hence maximizing the performance.

The average cumulative time to fetch RFCs is 195 milliseconds.

Advantages and Disadvantages:

Peer-to-peer networks are typically less secure than a centralized network because security is handled by the individual computers, not on the network. The resources of the computers in the network can become overburdened as they must support not only the workstation user, but also the requests from network users.

It is also difficult to provide different services because the desktop operating system typically used in this type of network is incapable of hosting the service.

Centralized networks have a higher initial setup cost. It is possible to set up a server on a desktop computer, but it is recommended that businesses invest in enterprise-class hardware and software. They also require a greater level of expertise to configure and manage the server hardware and software.