ECE 573 – INTERNET PROTOCOLS

PROJECT 1

Ankith Raja Divakar (araja2) & Madhumitha Raveendar (mraveen)

Objective:

1) Implement a simple peer-to-peer system with distributed index.
2) Analyse centralised file distribution system
3) Analyse P2P file distribution

Message formats:

a) Peer - RS Server:

The RS server is first started.

```
Registration Server is ready to receive
```

One peer is started, and it would try registering itself in the RS server by sending a registration request, ‘GET Register’ with its port number.

```
Sent request:
GET Register P2P-DI/1.0
ServerPort: 65440
Time: 2017-10-25 10:43:26
```

The RS server upon receiving the registration request would display the list of active peers.
The RS server would generate a cookie number for the peer and send back the cookie number in response.

```
Head node of peer list:
Node details: 10001, 192.168.56.1, 65440, True, 7200, 0, 2017-10-25 10:43:26.608741
Sent response:
P2P-DI/1.0 200 OK
cookie_id: 10001
Time: 2017-10-25 10:43:26
```

The peer would receive its unique cookie number and save it for any further correspondence with the RS server.

```
Received response:
P2P-DI/1.0 200 OK
cookie_id: 10001
Time: 2017-10-25 10:43:26
```

The peer would then query the RS server enquiring the other active peers registered with the RS server by sending the message, ‘GET PQuery’. As mentioned earlier, the peer includes its cookie number while contacting the RS server.

```
GET PQuery P2P-DI/1.0
Cookie_id: 10001
Time: 2017-10-25 10:43:26
```

The RS server would receive the PQuery request and respond with the list of active peers and their corresponding server port numbers. In this example, since only one peer is active, the RS server would respond with an empty list, thereby not sending the peer its own details.

```
Received request:
GET PQuery P2P-DI/1.0
Cookie_id: 10001
Time: 2017-10-25 10:43:26


10001
Sent response:
P2P-DI/1.0 200 OK
Time: 2017-10-25 10:43:26
```

The peer would receive an empty list from the RS server.

```
P2P-DI/1.0 200 OK
Time: 2017-10-25 10:43:26

Peer Details:
```

If another peer was started and it had queried the list of other active peers from RS server, it would have received the port number and IP address of the other active peer servers.

P2P-DI/1.0 200 OK
Time: 2017-10-25 13:09:58

Peer Details:
192.168.56.1, 65440

The following request would be sent to the RS server if a peer intends to leave a system.
Peer side:
GET Leave P2P-DI/1.0
Cookie_id: 10001
Time: 2017-10-25 13:37:17

RS server side:
Updated peer list:
Node details: 10001, 192.168.56.1, 65440, True, 7200, 0, 2017-10-25 13:37:17.628600
Sent response:
P2P-DI/1.0 200 OK
Time: 2017-10-25 13:37:17

Received request:
GET Leave P2P-DI/1.0
Cookie_id: 10001
Time: 2017-10-25 13:37:17

Updated peer list:
Node details: 10001, 192.168.56.1, 65440, False, 0, 0, 2017-10-25 13:37:17.628600
Sent response:
P2P-DI/1.0 200 OK
Time: 2017-10-25 13:37:17
The RS server upon receiving a LEAVE request would set the peer status to False, TTL is set to 0 from 7200, as hightlighted.

The following request would be sent to the RS server if the peer intends to KeepAlive.

Peer side:
GET KeepAlive P2P-DI/1.0
Cookie_id: 10001
Time: 2017-10-25 13:39:17

RS server side:

Updated peer list:
Node details: 10001, 192.168.56.1, 65440, True, 7199, 1, 2017-10-25 13:37:17.675145
Node details: 10002, 192.168.56.1, 65450, True, 7200, 1, 2017-10-25 13:38:34.720195
Sent response:
P2P-DI/1.0 200 OK
Time: 2017-10-25 13:38:34

Received request:
GET PQuery P2P-DI/1.0
Cookie_id: 10001
Time: 2017-10-25 13:39:17

10001
Sent response:
P2P-DI/1.0 200 OK
Time: 2017-10-25 13:39:17

Peer Details:
192.168.56.1, 65450

Received request:
GET KeepAlive P2P-DI/1.0
Cookie_id: 10001
Time: 2017-10-25 13:39:17

Updated peer list:
Node details: 10001, 192.168.56.1, 65440, True, 7200, 1, 2017-10-25 13:37:17.675145
Node details: 10002, 192.168.56.1, 65450, True, 7199, 1, 2017-10-25 13:38:34.720195
Sent response:
P2P-DI/1.0 200 OK
Time: 2017-10-25 13:39:17

It could be seen that, the TTL of the first peer (10001) is set to 7199 and the RS server upon receiving the KeepAlive request resets the TTL to 7200 for peer 1 as highlighted.

A peer which has left the system could re-register itself by sending a register request to the RS server along with its cookie id.

Peer side:
GET Register P2P-DI/1.0
Cookie_id: 10001
Host: LAPTOP-DG2UM7PK
Time: 2017-10-25 13:37:17

RS server side:
Updated peer list:
Node details: 10001, 192.168.56.1, 65440, False, 0, 0, 2017-10-25 13:37:17.628600
Sent response:
P2P-DI/1.0 200 OK
Time: 2017-10-25 13:37:17

Received request:
GET Register P2P-DI/1.0
Cookie_id: 10001
Host: LAPTOP-DG2UM7PK
Time: 2017-10-25 13:37:17

Updated peer list:
Node details: 10001, 192.168.56.1, 65440, True, 7200, 1, 2017-10-25 13:37:17.675145
Sent response:
P2P-DI/1.0 200 OK
Time: 2017-10-25 13:37:17

It can be observed that, when the RS receives a re-registration request, the status of the peer is set to True and the TTL is changed from 0 to 7200 and a variable, to denote how many times a peer has re-registered itself, is updated to 1 from 0.

In our program, KeepAlive and LEAVE requests are sent every 1 minute for testing purposes. These values are configurable to meet the requirements.

b) Peer – peer:

When a peer registers itself with RS server, the list of RFCs that it owns is built up.

RFC details: 7243, rfc7243.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 7321, rfc7321.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8117, rfc8117.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8144, rfc8144.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8154, rfc8154.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8217, rfc8217.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8218, rfc8218.txt, 192.168.56.1, 65440, 7200, Local, True

As per the requirement, it is mentioned that TTL of RFCs present in the peer itself is set to 7200. Status local indicates that the peer has the RFC. The peer is set to listen to other peers, if any.

When another peer registers itself with the RS server, it receives the peer list and queries the other registered peer for their RFC list.

Peer 2:

GET RFCQuery P2P-DI/1.0
Cookie_id: 10002
Time: 2017-10-25 16:46:43

Peer 1:

Peer is up and listining
client connedted ip:<('192.168.56.1', 56046)>
Received request:
GET RFCQuery P2P-DI/1.0
Cookie_id: 10002
Time: 2017-10-25 16:46:43

It is indicated to which port and IP address peer1 is connected.
Peer 1 receives the RFC request, RFCQuery and starts building RFC list to be sent to peer 2.

Peer 1:
Sent response:
P2P-DI/1.0 200 OK
Time: 2017-10-25 16:46:43

RFC Details:
7243, rfc7243.txt, 192.168.56.1, 65440, Remote
7321, rfc7321.txt, 192.168.56.1, 65440, Remote
8117, rfc8117.txt, 192.168.56.1, 65440, Remote
8144, rfc8144.txt, 192.168.56.1, 65440, Remote
8154, rfc8154.txt, 192.168.56.1, 65440, Remote
8217, rfc8217.txt, 192.168.56.1, 65440, Remote
8218, rfc8218.txt, 192.168.56.1, 65440, Remote
8219, rfc8219.txt, 192.168.56.1, 65440, Remote
8220, rfc8220.txt, 192.168.56.1, 65440, Remote
8221, rfc8221.txt, 192.168.56.1, 65440, Remote
8222, rfc8222.txt, 192.168.56.1, 65440, Remote

Here RFC status is set as Remote and sent to peer 2 to indicate that peer 2 has received the RFC list with the mentioned port number as the other peer's server, for the convenience of further queries.

Peer 2:
P2P-DI/1.0 200 OK
Time: 2017-10-25 16:46:43

RFC Details:
7243, rfc7243.txt, 192.168.56.1, 65440, Remote
7321, rfc7321.txt, 192.168.56.1, 65440, Remote
8117, rfc8117.txt, 192.168.56.1, 65440, Remote
8144, rfc8144.txt, 192.168.56.1, 65440, Remote
8154, rfc8154.txt, 192.168.56.1, 65440, Remote
8217, rfc8217.txt, 192.168.56.1, 65440, Remote
8218, rfc8218.txt, 192.168.56.1, 65440, Remote
8219, rfc8219.txt, 192.168.56.1, 65440, Remote
8220, rfc8220.txt, 192.168.56.1, 65440, Remote
8221, rfc8221.txt, 192.168.56.1, 65440, Remote
8222, rfc8222.txt, 192.168.56.1, 65440, Remote

Peer 2 receives the RFC list from peer 1 and merges it with its own RFC list.

Peer 2:

RFC details: 8219, rfc8219.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8220, rfc8220.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8221, rfc8221.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8222, rfc8222.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8223, rfc8223.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8228, rfc8228.txt, 192.168.56.1, 65440, 7200, Remote, True

In the RFC list of peer 2, the port number, IP address of the peer from which it had received the list is noted. Location status is set as remote to indicate that the peer doesn't own the RFC and the TTL of the file is set as 7200 which would keep decrementing and upon reaching 0, would set the status to false.

Peer 2 will then query for the RFCs that it doesn't have from peer 1.

Peer 2:
GET GetRFC P2P-DI/1.0
Cookie_id: 10002
Time: 2017-10-25 16:47:43

RFC Num:7321

Peer 1:

Received request:
GET GetRFC P2P-DI/1.0
Cookie_id: 10002
Time: 2017-10-25 16:46:43

RFC Num:7243

Peer 1 would receive the RFC request and send out the mentioned RFC file to peer 2.

Peer 2 would receive the RFC file that it had requested from peer 1.

Peer 2:
Sent response:
P2P-DI/1.0 200 OK
Time: 2017-10-25 16:46:43

C:\Users\madhu\Documents\NCSU_MS\ECE_573\Projects\Project_1\Files_2\RFCFiles\TestPath_1\rfc7243.txt
Total time to download just the files
0:00:00.067666

Here the location to which the file has been downloaded and the time taken to download the file is indicated.

The updated RFC list of peer 2 will be as follows:

RFC details: 7243, rfc7243.txt, 192.168.56.1, 65450, 7200, Local, True
RFC details: 7243, rfc7243.txt, 192.168.56.1, 65440, 7198, Remote, True
RFC details: 7321, rfc7321.txt, 192.168.56.1, 65450, 7200, Local, True
RFC details: 7321, rfc7321.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8117, rfc8117.txt, 192.168.56.1, 65450, 7200, Local, True
RFC details: 8117, rfc8117.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8144, rfc8144.txt, 192.168.56.1, 65450, 7200, Local, True
RFC details: 8144, rfc8144.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8154, rfc8154.txt, 192.168.56.1, 65450, 7200, Local, True
RFC details: 8154, rfc8154.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8217, rfc8217.txt, 192.168.56.1, 65450, 7200, Local, True
RFC details: 8217, rfc8217.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8218, rfc8218.txt, 192.168.56.1, 65450, 7200, Local, True
RFC details: 8218, rfc8218.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8219, rfc8219.txt, 192.168.56.1, 65450, 7200, Local, True
RFC details: 8219, rfc8219.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8220, rfc8220.txt, 192.168.56.1, 65450, 7200, Local, True
RFC details: 8220, rfc8220.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8221, rfc8221.txt, 192.168.56.1, 65450, 7200, Local, True
RFC details: 8221, rfc8221.txt, 192.168.56.1, 65440, 7200, Remote, True
RFC details: 8222, rfc8222.txt, 192.168.56.1, 65450, 7200, Local, True
RFC details: 8222, rfc8222.txt, 192.168.56.1, 65440, 7200, Remote, True

The RFC list will have the RFC list from peer 1 and its own RFC list merged together to be shared with other peers.

The following is the RFC list that peer 2 will share with other peers, indicating the files with peer 1 and peer 2 along with their corresponding IP address and port number.

7243, rfc7243.txt, 192.168.56.1, 65450, Remote
7243, rfc7243.txt, 192.168.56.1, 65440, Remote
7321, rfc7321.txt, 192.168.56.1, 65450, Remote
7321, rfc7321.txt, 192.168.56.1, 65440, Remote
8117, rfc8117.txt, 192.168.56.1, 65450, Remote
8117, rfc8117.txt, 192.168.56.1, 65440, Remote
8144, rfc8144.txt, 192.168.56.1, 65450, Remote
8144, rfc8144.txt, 192.168.56.1, 65440, Remote
8154, rfc8154.txt, 192.168.56.1, 65450, Remote
8154, rfc8154.txt, 192.168.56.1, 65440, Remote
8217, rfc8217.txt, 192.168.56.1, 65450, Remote
8217, rfc8217.txt, 192.168.56.1, 65440, Remote
8218, rfc8218.txt, 192.168.56.1, 65450, Remote
8218, rfc8218.txt, 192.168.56.1, 65440, Remote
8219, rfc8219.txt, 192.168.56.1, 65450, Remote
8219, rfc8219.txt, 192.168.56.1, 65440, Remote
8220, rfc8220.txt, 192.168.56.1, 65450, Remote
8220, rfc8220.txt, 192.168.56.1, 65440, Remote
8221, rfc8221.txt, 192.168.56.1, 65450, Remote
8221, rfc8221.txt, 192.168.56.1, 65440, Remote
8222, rfc8222.txt, 192.168.56.1, 65450, Remote
8222, rfc8222.txt, 192.168.56.1, 65440, Remote

In the meantime, peer 1 would have also queried peer 2 for its RFC list to update its own RFC list, which would be as follows:

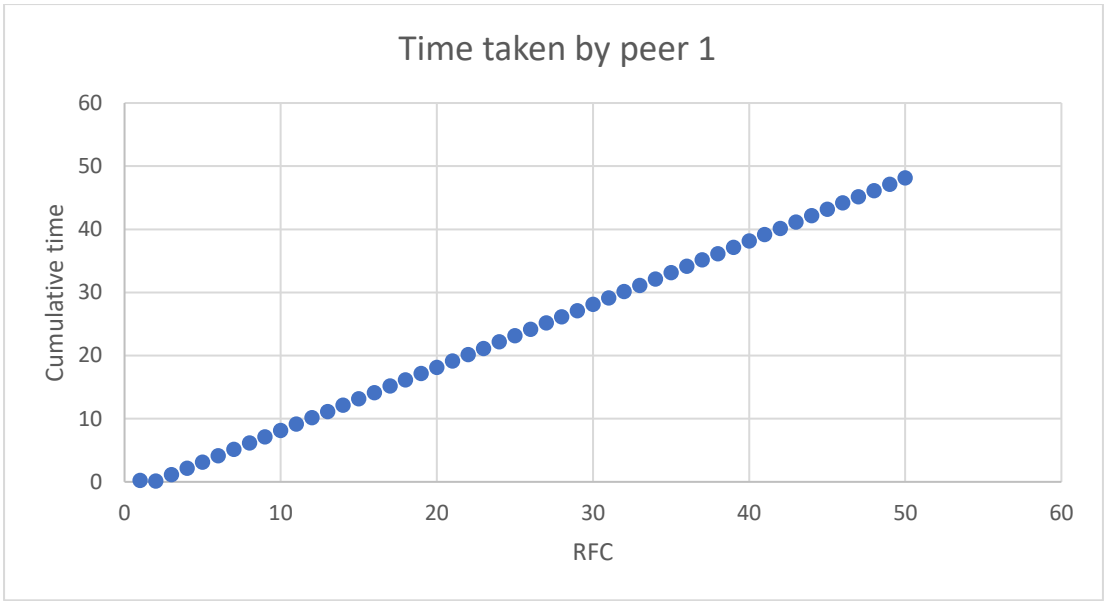
RFC details: 7243, rfc7243.txt, 192.168.56.1, 65450, 7200, Remote, True
RFC details: 7243, rfc7243.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 7321, rfc7321.txt, 192.168.56.1, 65450, 7200, Remote, True
RFC details: 7321, rfc7321.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8117, rfc8117.txt, 192.168.56.1, 65450, 7200, Remote, True

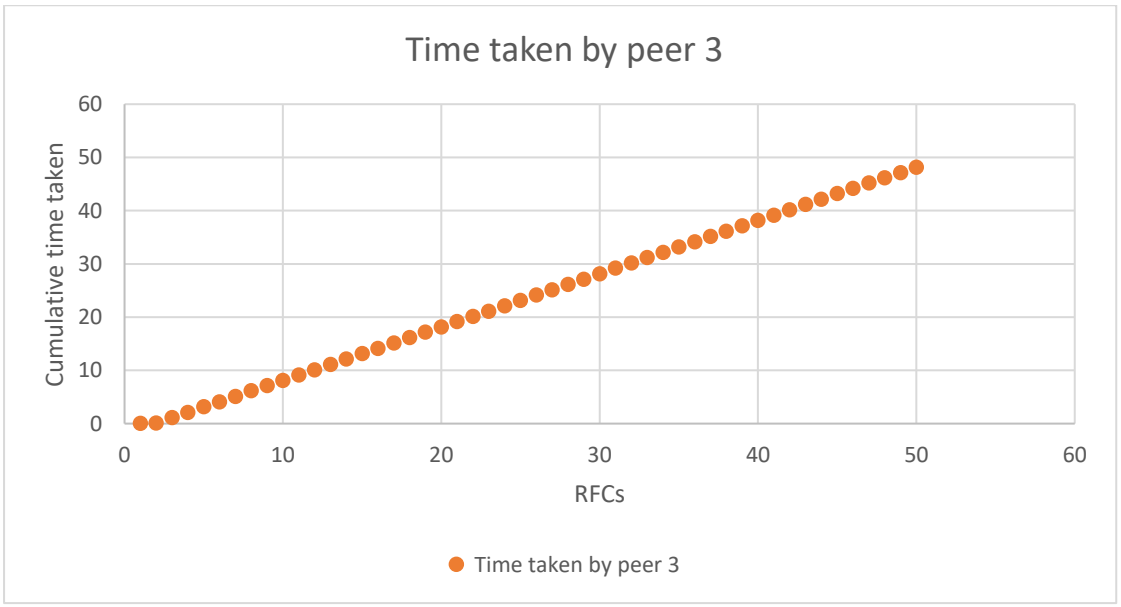
RFC details: 8117, rfc8117.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8144, rfc8144.txt, 192.168.56.1, 65450, 7200, Remote, True
RFC details: 8144, rfc8144.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8154, rfc8154.txt, 192.168.56.1, 65450, 7200, Remote, True
RFC details: 8154, rfc8154.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8217, rfc8217.txt, 192.168.56.1, 65450, 7200, Remote, True
RFC details: 8217, rfc8217.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8218, rfc8218.txt, 192.168.56.1, 65450, 7200, Remote, True
RFC details: 8218, rfc8218.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8219, rfc8219.txt, 192.168.56.1, 65450, 7200, Remote, True
RFC details: 8219, rfc8219.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8220, rfc8220.txt, 192.168.56.1, 65450, 7200, Remote, True
RFC details: 8220, rfc8220.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8221, rfc8221.txt, 192.168.56.1, 65450, 7200, Remote, True
RFC details: 8221, rfc8221.txt, 192.168.56.1, 65440, 7200, Local, True
RFC details: 8222, rfc8222.txt, 192.168.56.1, 65450, 7200, Remote, True
RFC details: 8222, rfc8222.txt, 192.168.56.1, 65440, 7200, Local, True

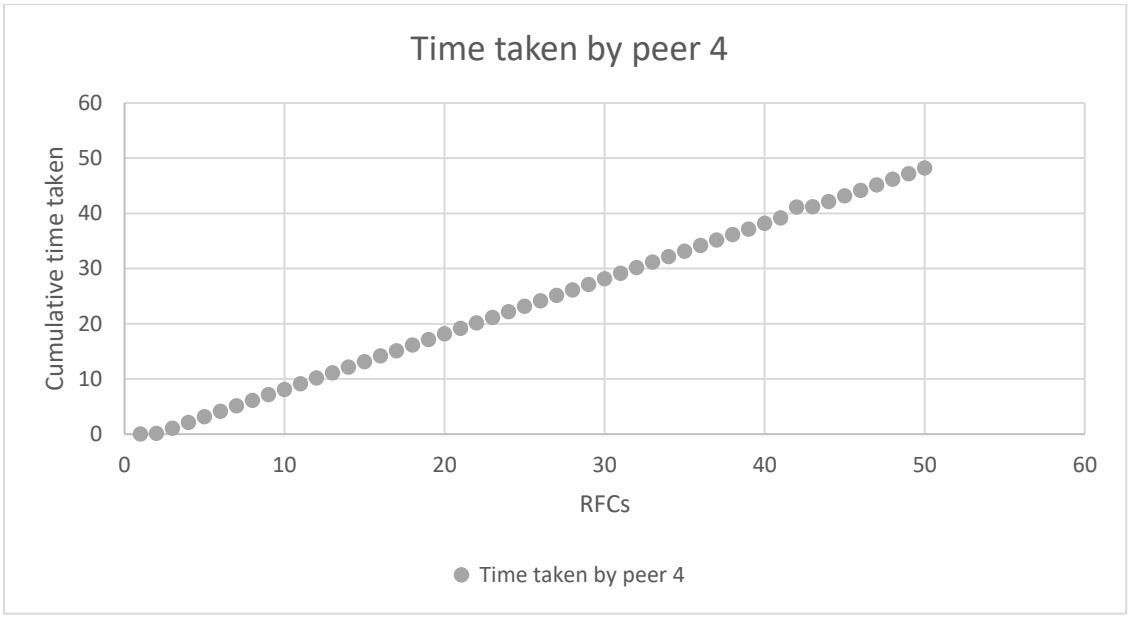
For testing purposes, GetRFC request is sent out every 1 second and RFCQuery is sent out every 60 seconds to all the peers. These values can be calibrated to meet the requirements.

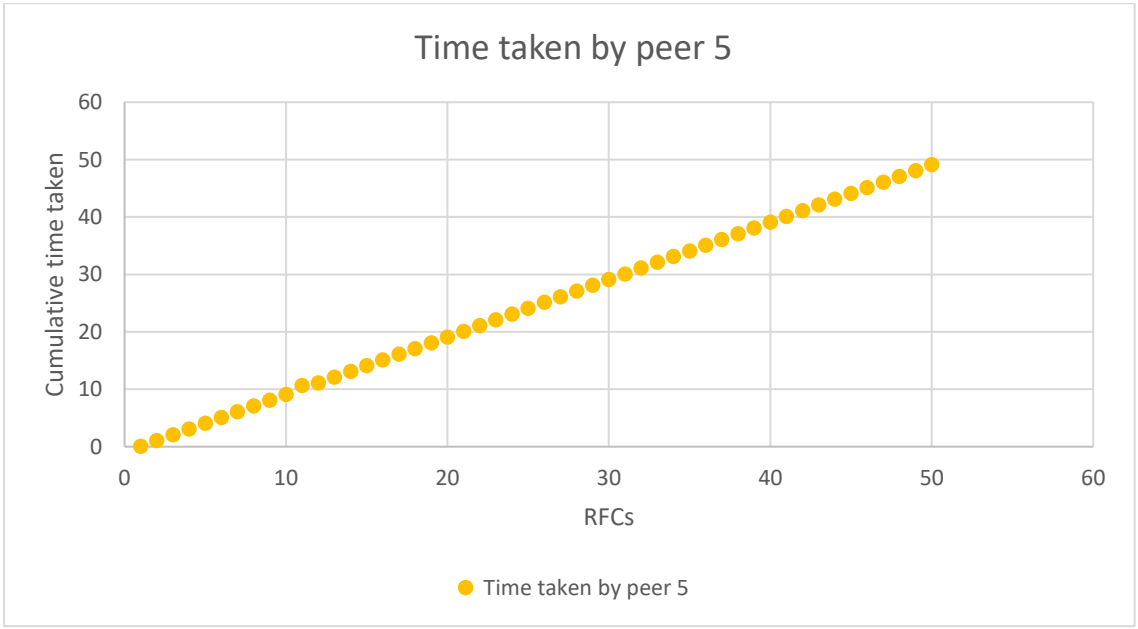
Task 1:

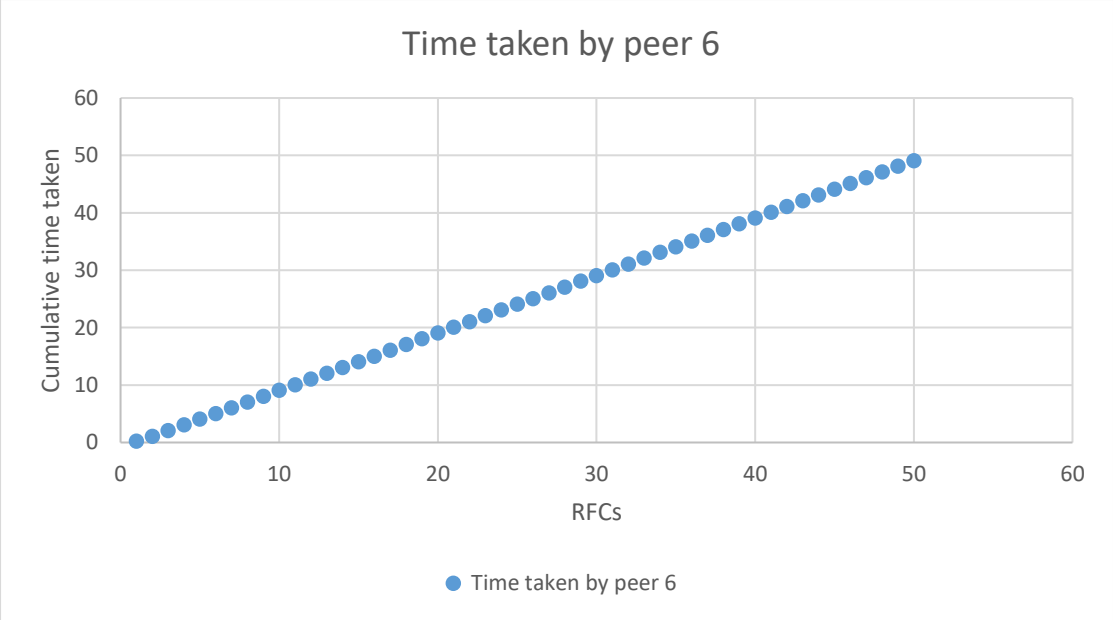
In the first step, all six peers register with the RS and receive the peer list. In the second step, peers P1, . . . , P5 query P0 and obtain its RFC index; clearly, the index simply contains the 60 RFC files that peer P0 holds. In the third step, each of the peers P1, . . . , P5 starts a loop to download 50 RFC files from peer P0, one file at a time. Each peer also records the time it takes to download each RFC file from peer P0, as well as the cumulative time. You are to plot the cumulative download time against the number of RFCs for each peer P1, . . . , P5. Note that this operation emulates a centralized server (i.e., P0) that maintains all the information and handles simultaneous requests from clients (i.e., P1, . . . , P5).


Time taken by peer 1
Cumulative time
RFC
0
10
20
30
40
50
60


Time taken by peer 3
Cumulative time taken
RFCs
Time taken by peer 3
0
10
20
30
40
50
60

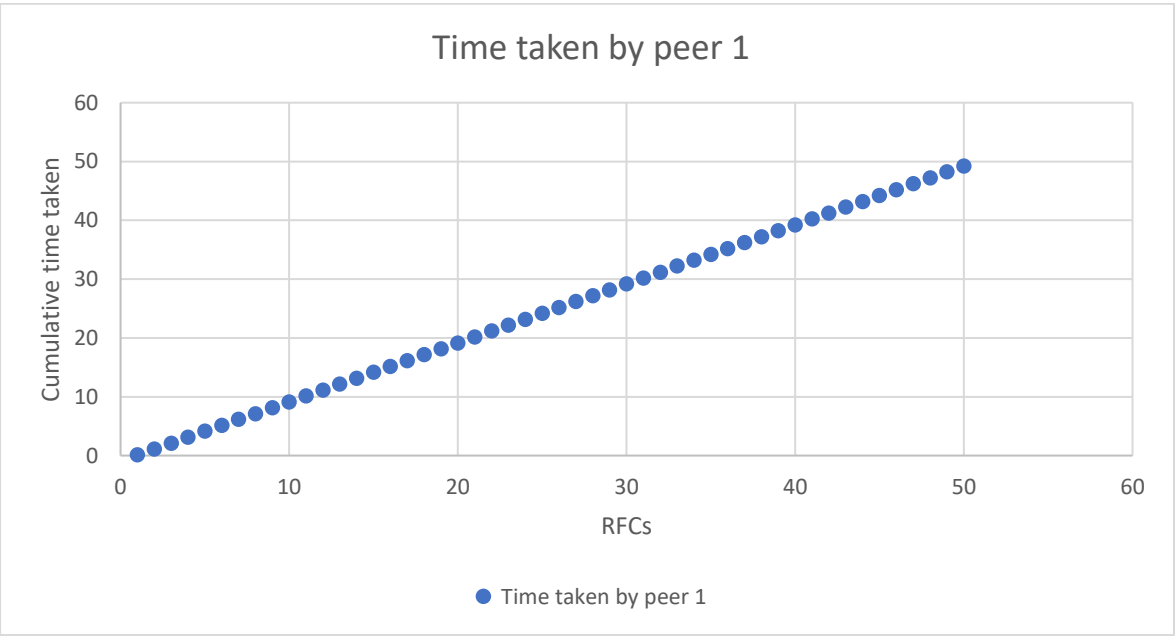

Time taken by peer 4
Cumulative time taken
60
50
40
30
20
10
0
0
10
20
30
40
50
60
RFCs
Time taken by peer 4

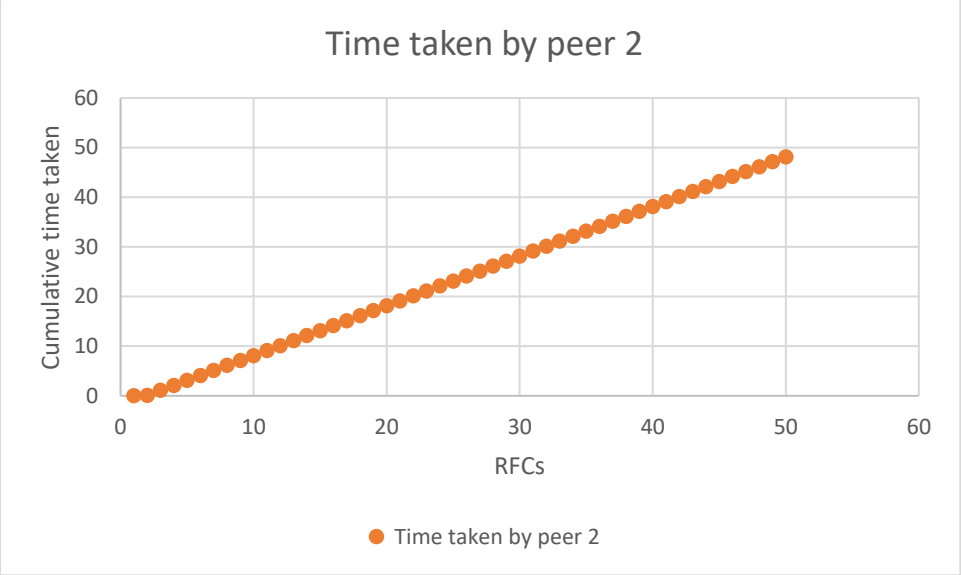

Time taken by peer 5
Cumulative time taken
60
50
40
30
20
10
0
0
10
20
30
40
50
60
RFCs
Time taken by peer 5

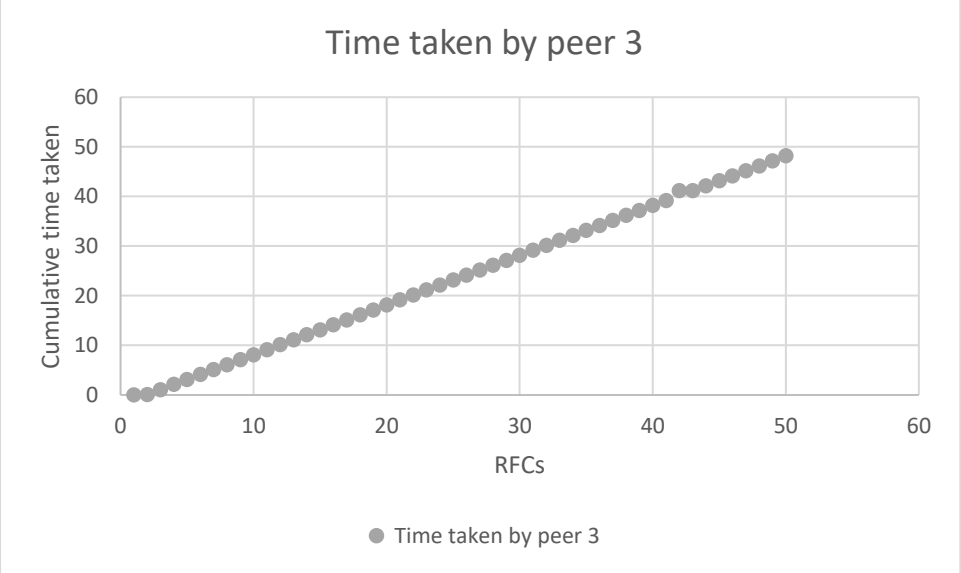


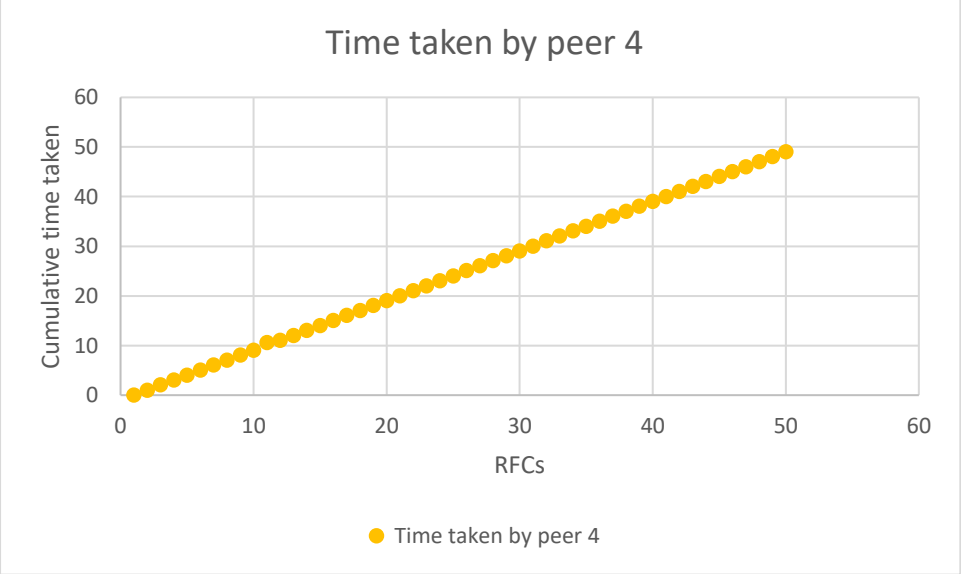
Task 2:

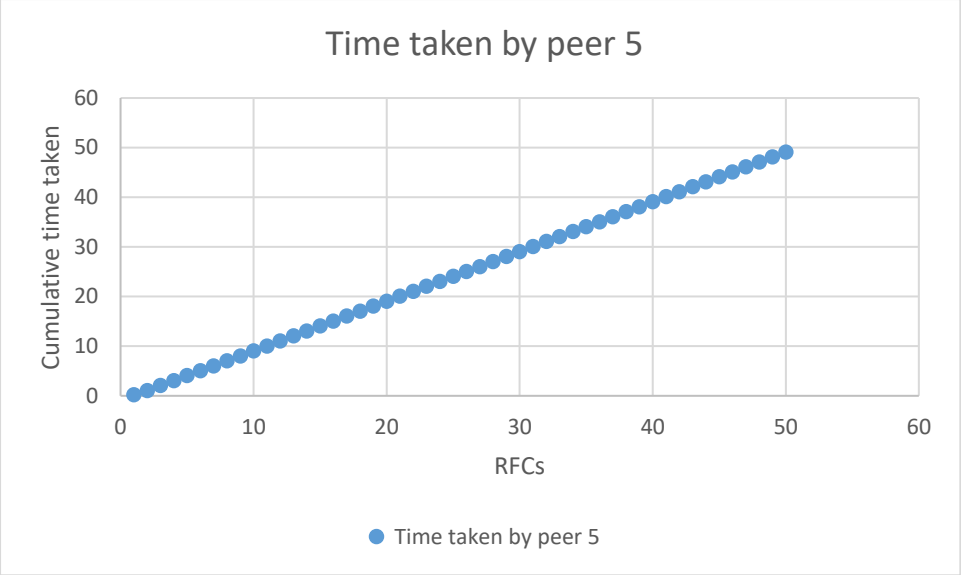
In the first step, all six peers register with the RS and obtain the peer list. In the second step, each of the six peers queries the other five peers and obtains their RFC index which it merges with its own; at the end of this step, each peer will have the whole RFC index containing 60 RFC files which are evenly distributed among the peers. In the third step, each of the six peers starts a loop to download the 50 RFC files it does not have from the corresponding remote peer. You are again to plot the cumulative download time against the number of RFCs for each of the six peers. It is not difficult to see that there is a best-case and worst-case scenario in this operation; implement both and provide download time figures for both.

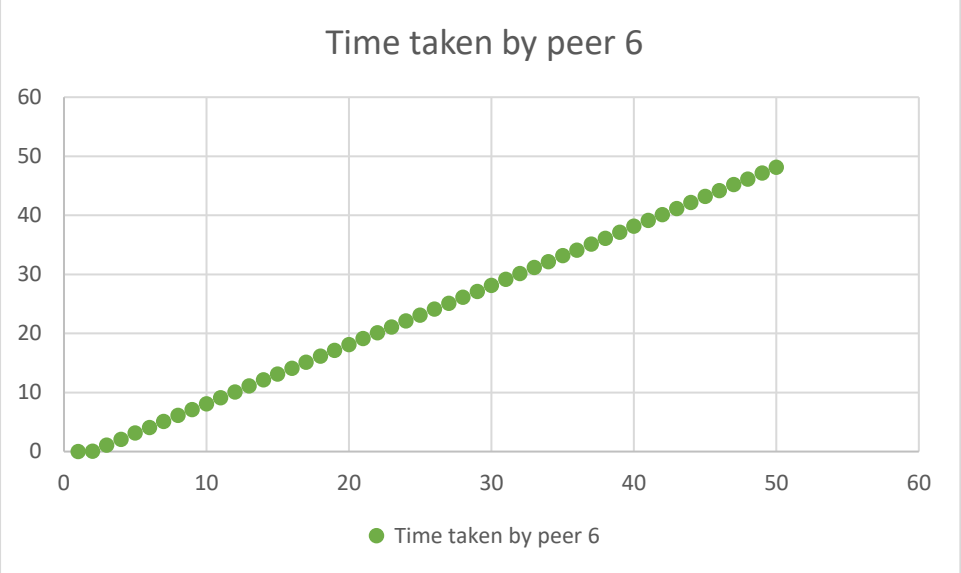



Time taken by peer 2
Cumulative time taken
0
10
20
30
40
50
60
RFCs
Time taken by peer 2


Time taken by peer 3
Cumulative time taken
0
10
20
30
40
50
60
RFCs
Time taken by peer 3


Time taken by peer 4
Cumulative time taken
0
10
20
30
40
50
60
RFCs
Time taken by peer 4





Comparison of centralised and decentralised system:

In centralised system, one peer must serve all the other peers. On the other hand, in a decentralised system, peers serve each other, thereby ensuring a more balanced system. This will ensure a lesser time consumption for all the peers to receive all the peers. The decentralised approach ensures robustness as there is no single point of failure unlike a centralised approach, wherein if the source peer for file transfer fails, no other peer would receive the files. On the other hand, in the decentralised system, even though one peer has failed, the other peers can receive files from all the other peers.

The graph plotted for both Task 1 and Task 2 is similar as the peer to peer system is designed to limit user inputs, due to which the system uses timers and threads to download the RFC and threads run every one second. This is causing the graph to be linear there by we are not able to notice the timing advantage of the distributed system on the graph plotted.

Conclusion:

A simple peer to peer system has been successfully implemented. A comparison has been drawn between distributed and centralised systems.