Internet Protocols

CSC/ECE 573

Project 1

Group:
Madhura Bhide 200201405
Sayali Kamat 200202513

A. OBJECTIVE

1)To implement Peer-to-Peer with Distributed Index (P2P-DI) System for Downloading RFCs
Internet protocol standards are defined in documents called “Requests for Comments” (RFCs).
2)To build a P2P-DI system in which peers who wish to download an RFC that they do not have in their hard drive, may download it from another active peer who does over TCP.
3)To compare Client-Server and Peer-to-Peer architecture in terms of cumulative download time and scalability.

B. MESSAGE FORMATS

PEER TO RS COMMUNICATION

Peer registers to RS

Sample Format: <method><sp><P2P-DI>/<P2P version><sp>Host:*host*<sp>OS: <OSversion><el>
Example: Peer A registers to RS.
Register<sp>P2P-DI/1.0<sp>Host:HostA<sp>OS: Mac OS 10.4.1<el>

RS sends cookie to peer

Sample Format: <method><sp><P2P-DI>/<P2P version><sp>OS:
<OSversion><el>Cookie:<cookie_number><el>
Example: RS send Register response with cookie to Peer A
Register<sp>P2P-DI/1.0<sp>OS: Mac OS 10.4.1<el>Cookie:1<el>

Peer sends PQuery to get list of active Peers

Sample Format: <method><sp><P2P-DI>/<P2P version><sp>OS: <OSversion><el>
Cookie:<cookie_number><el>
Example: Host A sends PQuery to RS to get the list of active peers.
PQuery<sp>P2P-DI/1.0<sp>OS: Mac OS 10.4.1<el>Cookie:1<el>

RS sends PQresponse containing list of active Peers

Sample Format: <method><sp><P2P-DI>/<P2P version><sp>OS: <OSversion><el>
Host:host_number<sp>IP:IP_number<sp>RFCserver:RFC_Portnumber<el>Cookie:<cookienumber><el>
Example: RS sends PQresponse to Host A.
PQ_Response<sp>P2P-DI/1.0 <sp>OS: Mac OS 10.4.1<el>Host:HostA<sp>IP:192.168.1.2<sp>RFCserver:1000<el>

Peer sends keepalive to RS after every 2 seconds

Sample Format: <method><sp><P2P-DI>/<P2P version><sp>OS: <OSversion><el>
Cookie:<cookienumber><el>
Example: Host A sends keepalive messages after every 2 seconds.
Keepalive<sp>Host:hostA<sp>Cookie:5<el>

Peer sends leave message to RS

Sample Format: <method><sp><P2P-DI>/<P2P version><sp>OS: <OSversion><el>
Cookie:<cookie_number><el>
Example: Host with cookie 1 sends leave message to RS.
Leave<sp>P2P-DI/1.0<sp>OS: Mac OS 10.4.1<el>Cookie:1<el>

```
RS socket listening on port number 65423!
connection from ('192.168.0.43', 58530)
Register<sp>P2P-DI/1.0<sp>Host:hostB<sp>OS: Mac OS 10.4.1<el>
Registration successful!
Updated Register:
{1: ['hostB', True, 7200, 1000, 1, '2017-10-24', '192.168.0.43']}
connection from ('192.168.0.43', 58531)
Keepalive<sp>Host:hostB<sp>Cookie:1<el>
connection from ('192.168.0.43', 58534)
Register<sp>P2P-DI/1.0<sp>Host:hostA<sp>OS: Mac OS 10.4.1<el>
Registration successful!
Updated Register:
{1: ['hostB', True, 7200, 1000, 1, '2017-10-24', '192.168.0.43'], 2: ['hostA', True, 7200, 500, 1, '2017-10-24', '192.168.0.43']}
```

```
TTL for:hostB
['hostB', True, 7140, 1000, 1, '2017-10-24', '192.168.0.43']
connection from ('192.168.0.43', 58655)
Keepalive<sp>Host:hostA<sp>Cookie:2<el>
TTL for:hostA
['hostA', True, 7140, 500, 1, '2017-10-24', '192.168.0.43']
```

PEER TO PEER COMMUNICATION

Peer client sends RFCQuery to RFC server on active Peer

Sample Format: <method><sp><P2P-DI>/<P2P version><el>
Example: RFCQuery<sp>P2P-DI/1.0<el>

RFC server sends RFCResponse to Peer containing RFC Index

Sample Format: RFCResponse<sp>P2P-DI/1.0<el><object List of RFCIndex>
Example:RFCResponse<sp>P2P-DI/1.0<el><{7501:[‘7501.pdf’,’hostA’,7200], 7502:[‘7502.pdf’,’host B’,7180]}

Peer client sends GetRFC to RFC server

Sample Format: <method><sp><P2P-DI>/<P2P version><el>
Example: GetRFC<sp>P2P-DI/1.0<el>

```
C:\Users\Madhura\Desktop\IP Project>py Trial2.py
connecting to 192.168.0.44 port 65423
closing Register socket close
Got Cookie:1
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.44 port 65423
closing PQ socket close
PQ_Response<sp>P2P-DI/1.0 <sp>OS: Mac OS 10.4.1<el>Host:hostB<sp>IP:192.168.0.43<sp>RFCPort:1000<el>Host:hostA<sp>IP:192.168.0.43<sp>RFCPort:500<el>
starting up on 192.168.0.43 port 1000
waiting for a connection
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.44 port 65423
connection from ('192.168.0.43', 58606)
waiting for a connection
Clean up the connection
connection from ('192.168.0.43', 58607)
waiting for a connection
Done sending
connection from ('192.168.0.43', 58608)
waiting for a connection
Clean up the connection
connection from ('192.168.0.43', 58609)
waiting for a connection
Done sending
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.44 port 65423
```

```
C:\Users\Madhura\Desktop\IP Project>py Trial1.py
connecting to 192.168.0.44 port 65423
closing Register socket close
Got Cookie:2
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.44 port 65423
closing PQ socket close
PQ_Response<sp>P2P-DI/1.0 <sp>OS: Mac OS 10.4.1<el>Host:hostB<sp>IP:192.168.0.43<sp>RFCPort:1000<el>Host:hostA<sp>IP:192.168.0.43<sp>RFCPort:500<el>
starting up on 192.168.0.43 port 500
waiting for a connection
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.44 port 65423
connecting to 192.168.0.43 port 1000
received "{7552: ['7552.pdf', 'hostB', 7200], 7553: ['7553.pdf', 'hostB', 7200], 7554: ['7554.pdf', 'hostB', 7200], 7555: ['7555.pdf', 'hostB', 7200], 7556: ['7556.pdf', 'h
ostB', 7200], 7557: ['7557.pdf', 'hostB', 7200], 7558: ['7558.pdf', 'hostB', 7200], 7559: ['7559.pdf', 'hostB', 7200], 7560: ['7560.pdf', 'hostB', 7200], 7501: ['7501.pdf',
 'hostB', 7200], 7502: ['7502.pdf', 'hostB', 7200], 7505: ['7505.pdf', 'hostB', 7200], 7506: ['7506.pdf', 'hostB', 7200], 7507: ['7507.pdf', 'hostB', 7200], 7508: ['7508.pd
f', 'hostB', 7200], 7509: ['7509.pdf', 'hostB', 7200], 7510: ['7510.pdf', 'hostB', 7200], 7511: ['7511.pdf', 'hostB', 7200], 7512: ['7512.pdf', 'hostB', 7200], 7513: ['7513
.pdf', 'hostB', 7200], 7514: ['7514.pdf', 'hostB', 7200], 7515: ['7515.pdf', 'hostB', 7200], 7516: ['7516.pdf', 'hostB', 7200], 7517: ['7517.pdf', 'hostB', 7200], 7518: ['7
518.pdf', 'hostB', 7200], 7519: ['7519.pdf', 'hostB', 7200], 7520: ['7520.pdf', 'hostB', 7200], 7521: ['7521.pdf', 'hostB', 7200], 7522: ['7522.pdf', 'hostB', 7200], 7523:
['7523.pdf', 'hostB', 7200], 7524: ['7524.pdf', 'hostB', 7200], 7525: ['7525.pdf', 'hostB', 7200], 7526: ['7526.pdf', 'hostB', 7200], 7527: ['7527.pdf', 'hostB', 7200], 752
8: ['7528.pdf', 'hostB', 7200], 7529: ['7529.pdf', 'hostB', 7200], 7530: ['7530.pdf', 'hostB', 7200], 7531: ['7531.pdf', 'hostB', 7200], 7532: ['7532.pdf', 'hostB', 7200],
7533: ['7533.pdf', 'hostB', 7200], 7534: ['7534.pdf', 'hostB', 7200], 7535: ['7535.pdf', 'hostB', 7200], 7536: ['7536.pdf', 'hostB', 7200], 7537: ['7537.pdf', 'hostB', 7200
], 7538: ['7538.pdf', 'hostB', 7200], 7539: ['7539.pdf', 'hostB', 7200], 7540: ['7540.pdf', 'hostB', 7200], 7541: ['7541.pdf', 'hostB', 7200], 7542: ['7542.pdf', 'hostB', 7
200], 7543: ['7543.pdf', 'hostB', 7200], 7544: ['7544.pdf', 'hostB', 7200], 7545: ['7545.pdf', 'hostB', 7200], 7546: ['7546.pdf', 'hostB', 7200], 7547: ['7547.pdf', 'hostB'
, 7200], 7548: ['7548.pdf', 'hostB', 7200], 7549: ['7549.pdf', 'hostB', 7200], 7550: ['7550.pdf', 'hostB', 7200], 7551: ['7551.pdf', 'hostB', 7200]}"
closing socket
received "28446"
file opened
receiving data...
Successfully get the file
connection closed
connection closed
```

Code details:

Submitted code contains 5 .py files:
1. RS Code (Registration Server)
2. Register code (Peer client method for register)
3. PQuery code (Peer client method for PQuery)
4. Leave code (Peer client method for Leave)
5. Peer_Client1.py (Client methods for client part of Peer to call)
6. Main Peer code (Containing Server thread which is always in listening mode and file download calls which call methods specified in Peer_Client1.py)

Data.txt file:
Sample file for assigning IPs and Ports to Peers

MakeFile.txt :
Instructions for running Python code

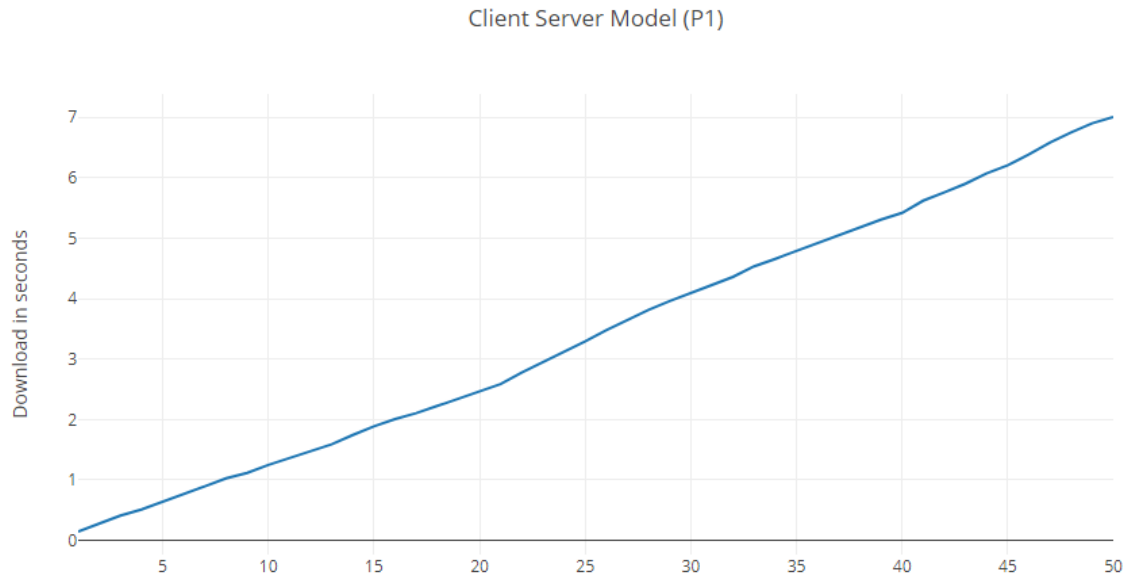
We have used 'dictionary' data structure in Python for storing each Peer's RFC Index. Keys of this dictionary contains RFC number and value contains RFC name (String), hostname of Peer containing that RFC (String) and Time To Leave(TTL) value (Integer)

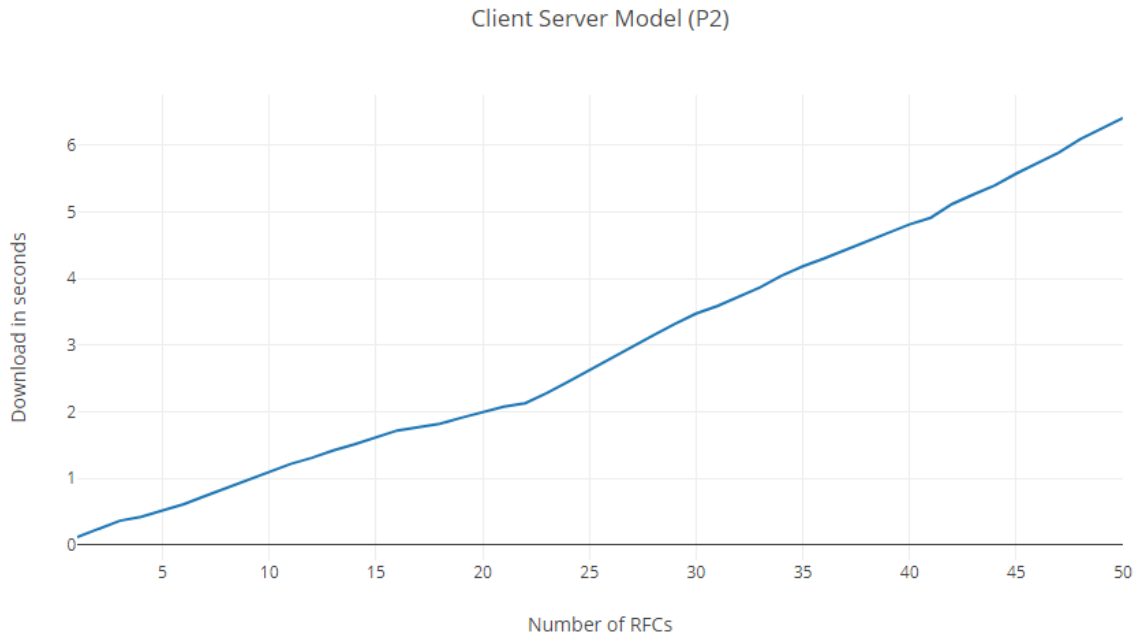
Example: {7501:['7501.pdf', 'hostA',7200]}

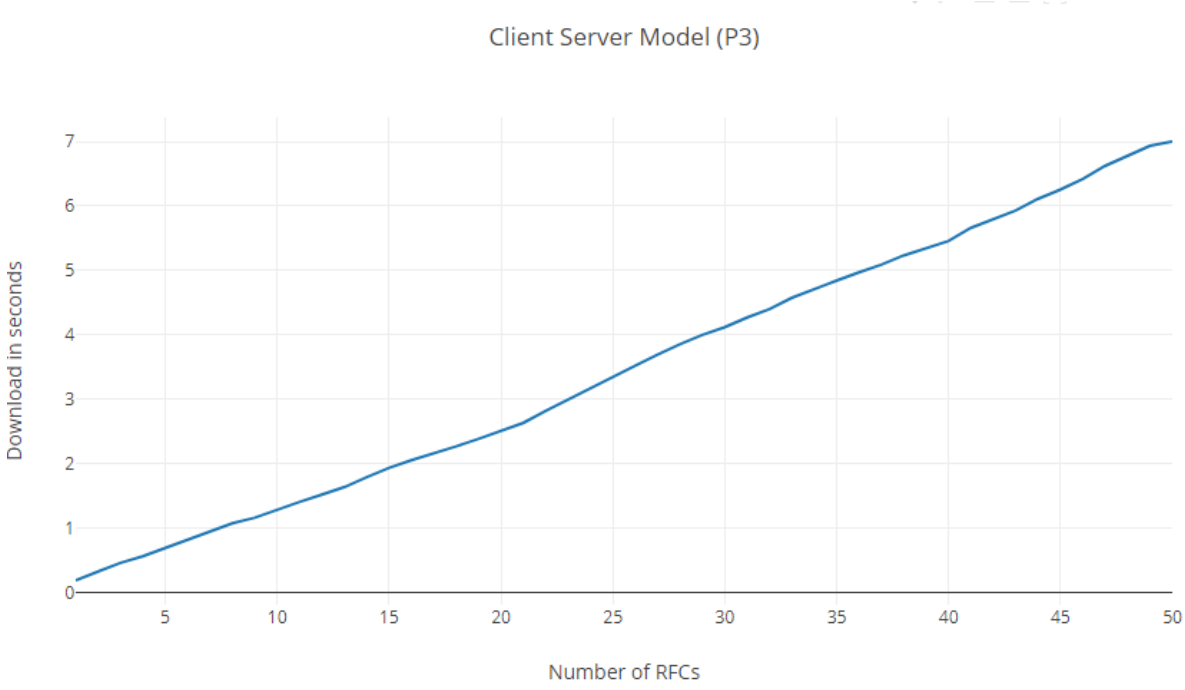
C. The download time curves for Task 1 (Client Server Model)

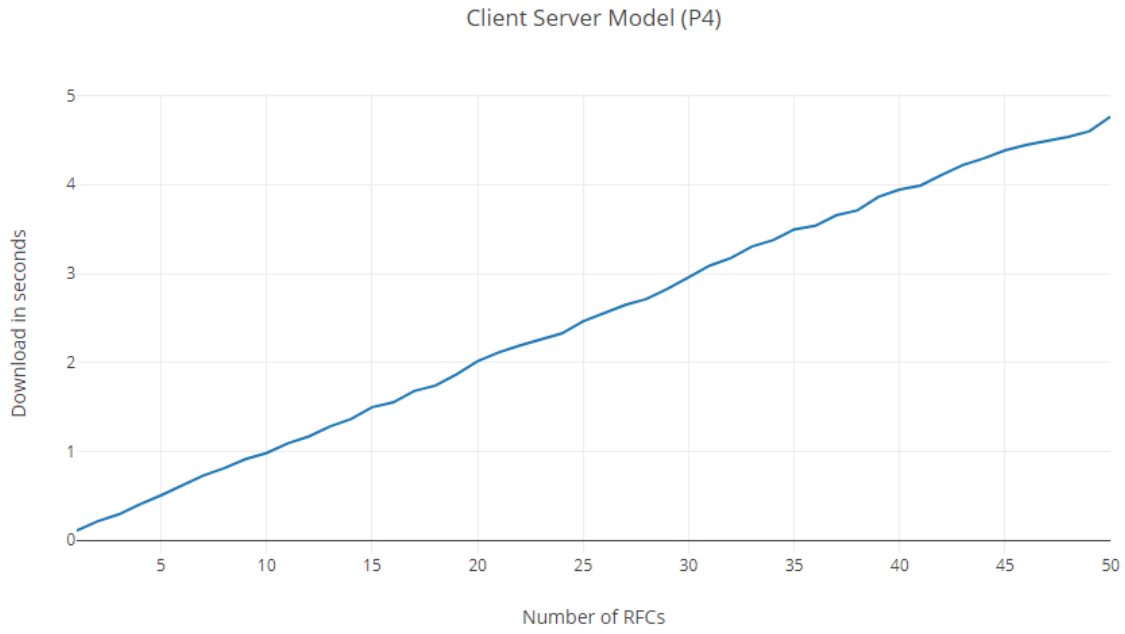
Scenario:

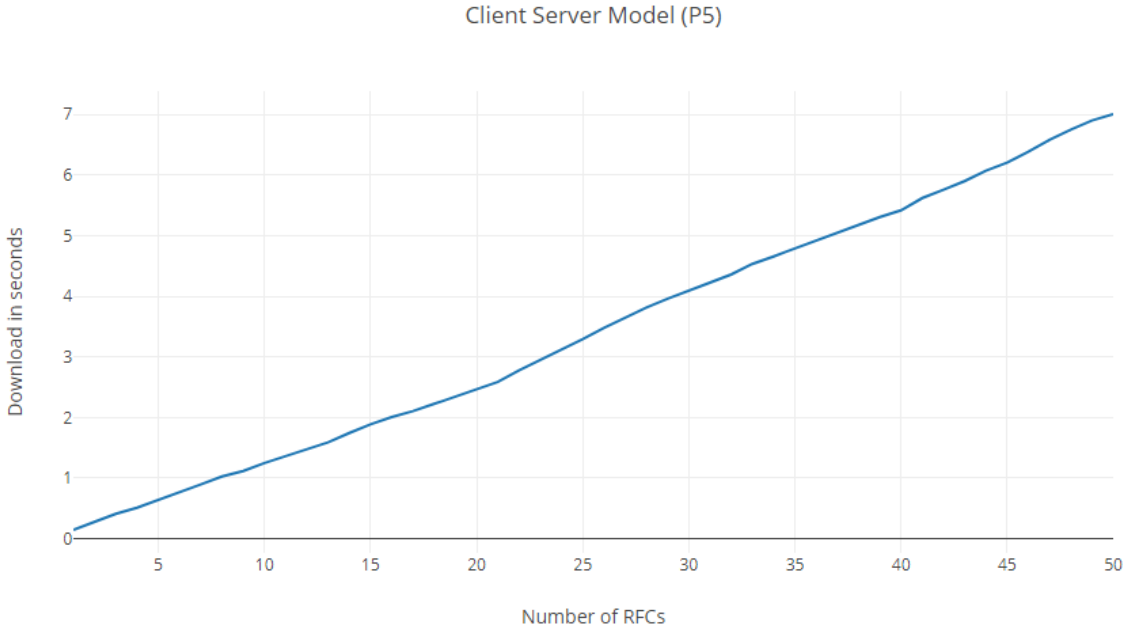
Here, we have created 6 peers P0, P1,P2,P3,P4 and P5 with P0 having 60 recent RFCs. All the 6 peers are implemented on 6 window's terminals while RS is implemented on another terminal on another laptop. Next, peers query RS and gets RFC index. Each of the peers P1 to P6 starts a loop to download 50 RFC files from peer P0, one file at a time. Each peer also records the time it takes to download from P0. Following are the graphs for each peer.






Client Server Model (P3)
Download in seconds
0
1
2
3
4
5
6
7
5
10
15
20
25
30
35
40
45
50
Number of RFCs


Client Server Model (P4)
Download in seconds
0
1
2
3
4
5
5
10
15
20
25
30
35
40
45
50
Number of RFCs


Client Server Model (P5)
Download in seconds
0
1
2
3
4
5
6
7
5
10
15
20
25
30
35
40
45
50
Number of RFCs

D. The download time curves for Task 2 (peer to peer model)

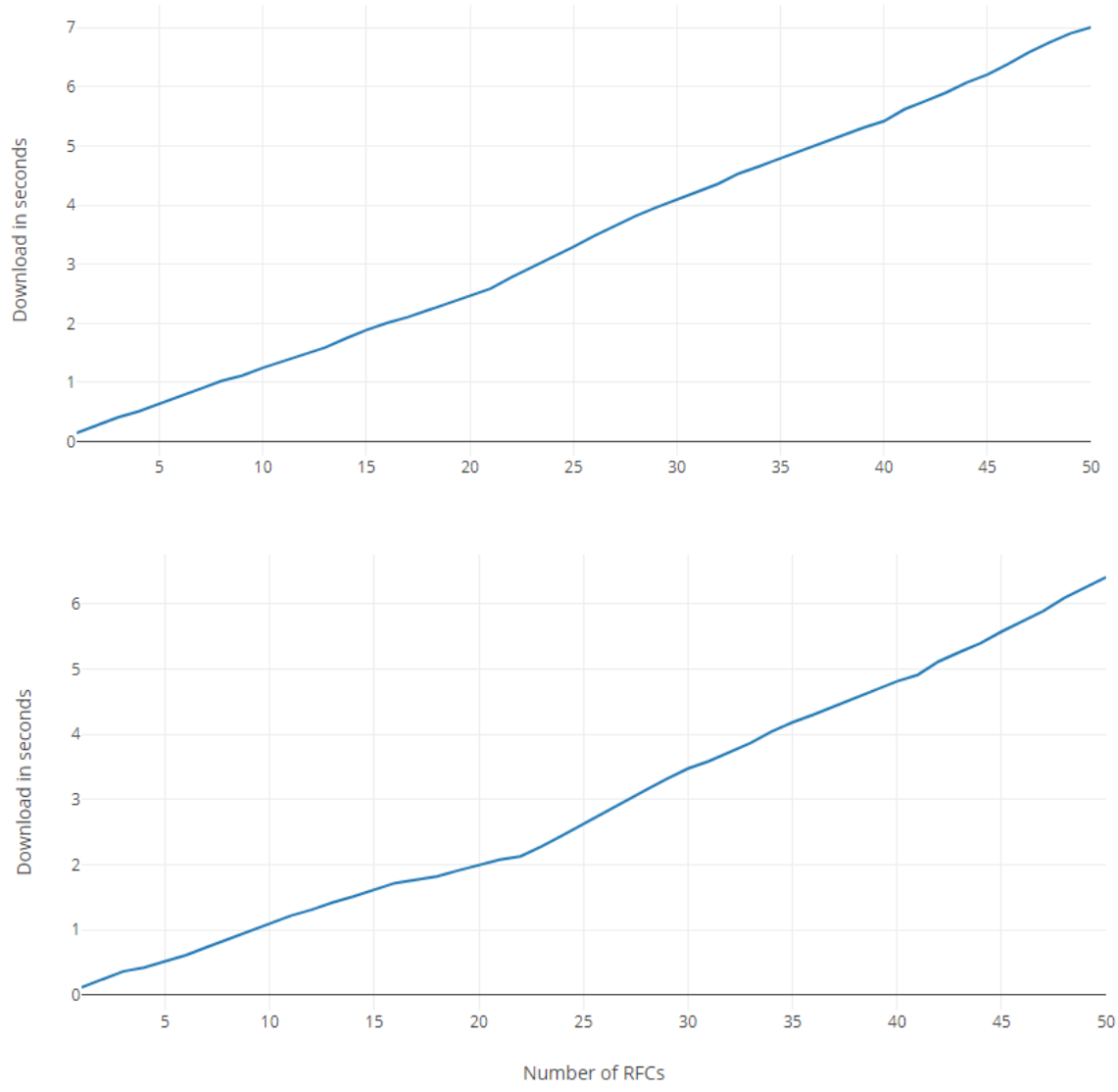
Scenario:

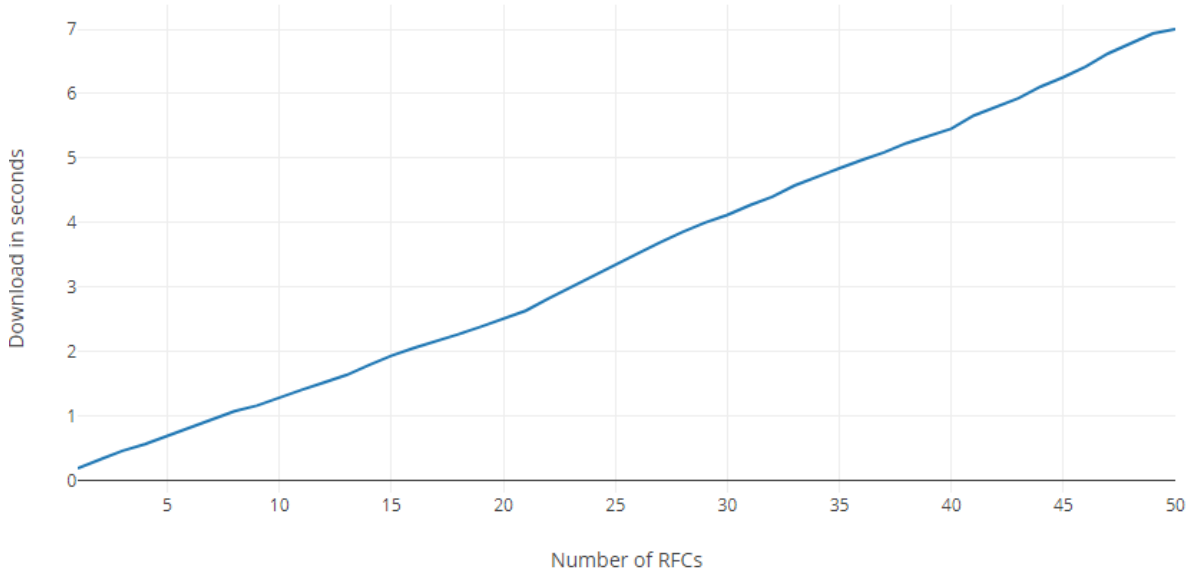
Here, we have created 6 peers P0, P1, P2, P3, P4 and P5 with each having 10 recent RFCs. All the 6 peers are implemented on 6 window's terminals while RS is implemented on another terminal. Next, peers query RS and gets RFC index. Each of the peers P0 to P6 starts a loop to download 50 RFC files from other peers, one file at a time. Each peer also records the time it takes to download.

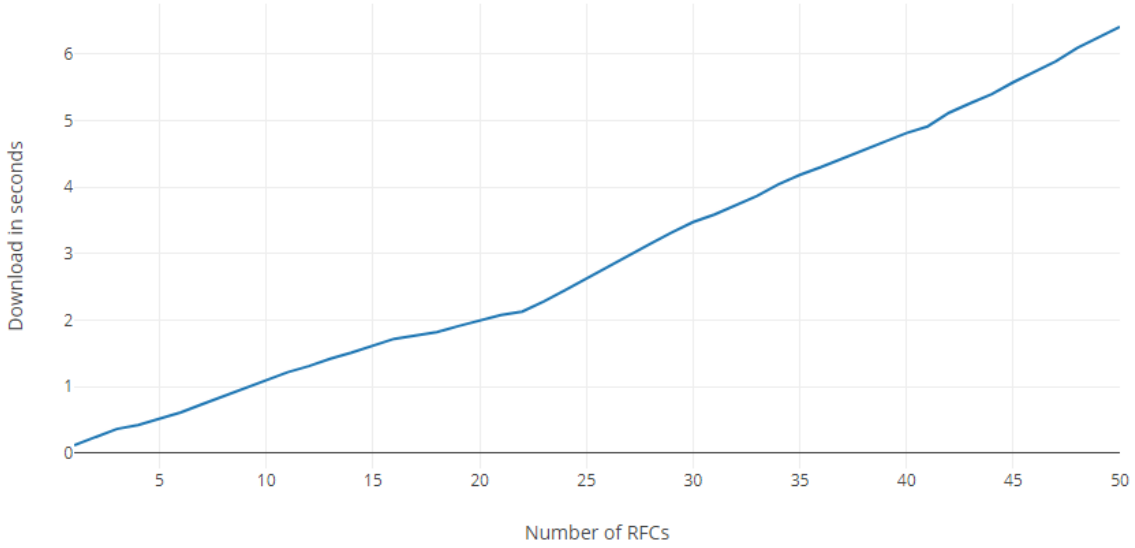
Best case scenario and worst-case scenario analysis:

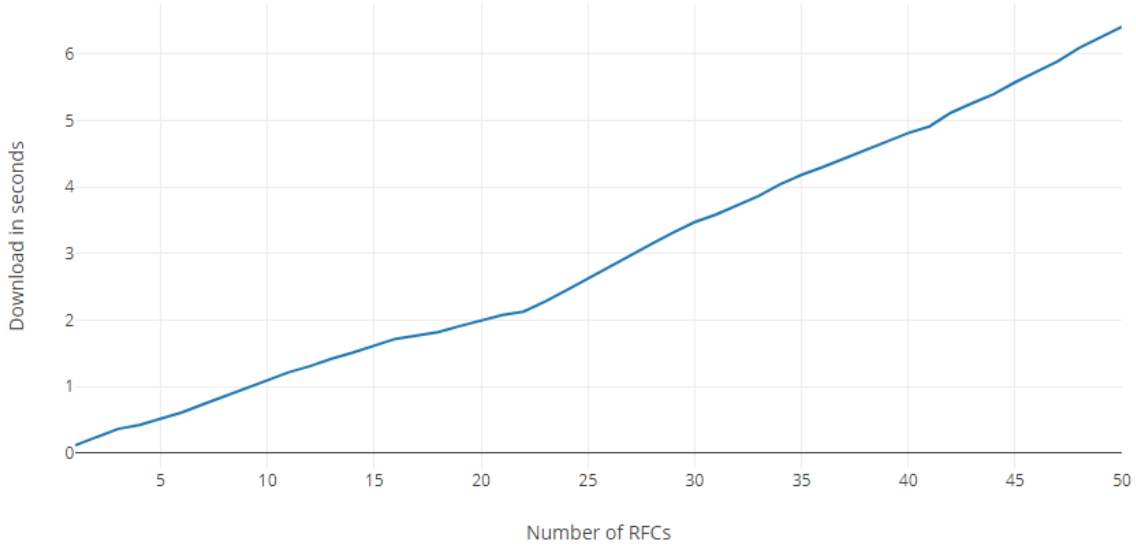
Best case scenario for any protocol or algorithm depends upon input or the way algorithms handles different set of input. In this case, our best and worst case will depend upon the type of input. The best-case scenario in terms of less download time for files will be if while querying the first peers has all the RFC files needed, then the peer can download all the files in one go from one peer. While, if the files are distributed evenly, the peer needs to query others peer to get RFCs, this will add processing delay and waiting delay to the total download time.

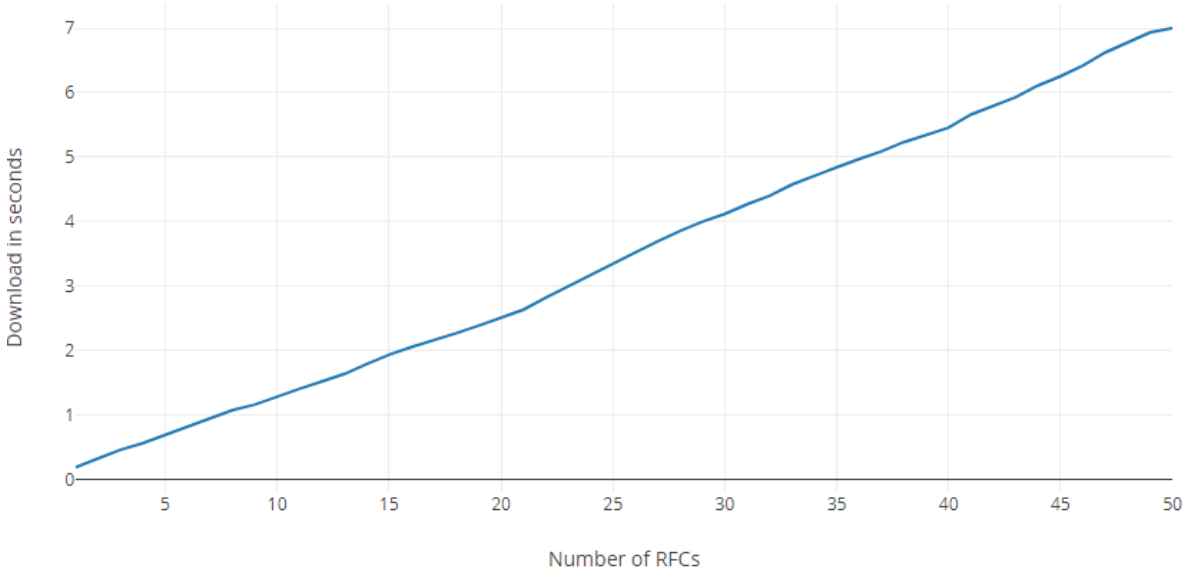
Best case scenario:



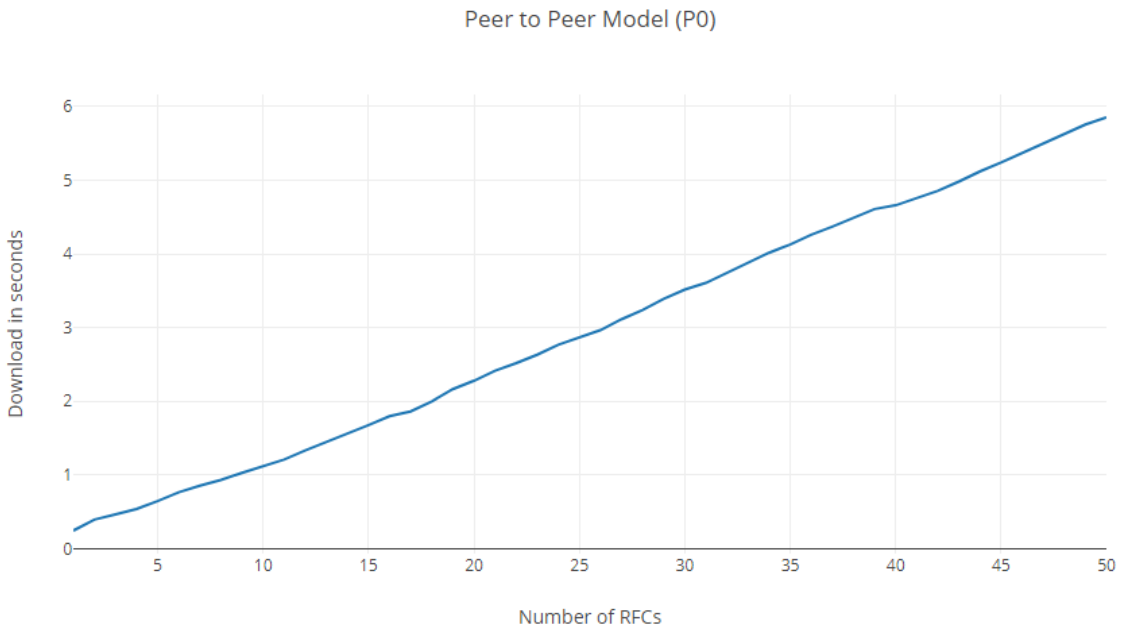

Download in seconds
0
1
2
3
4
5
6
7
5
10
15
20
25
30
35
40
45
50
Number of RFCs

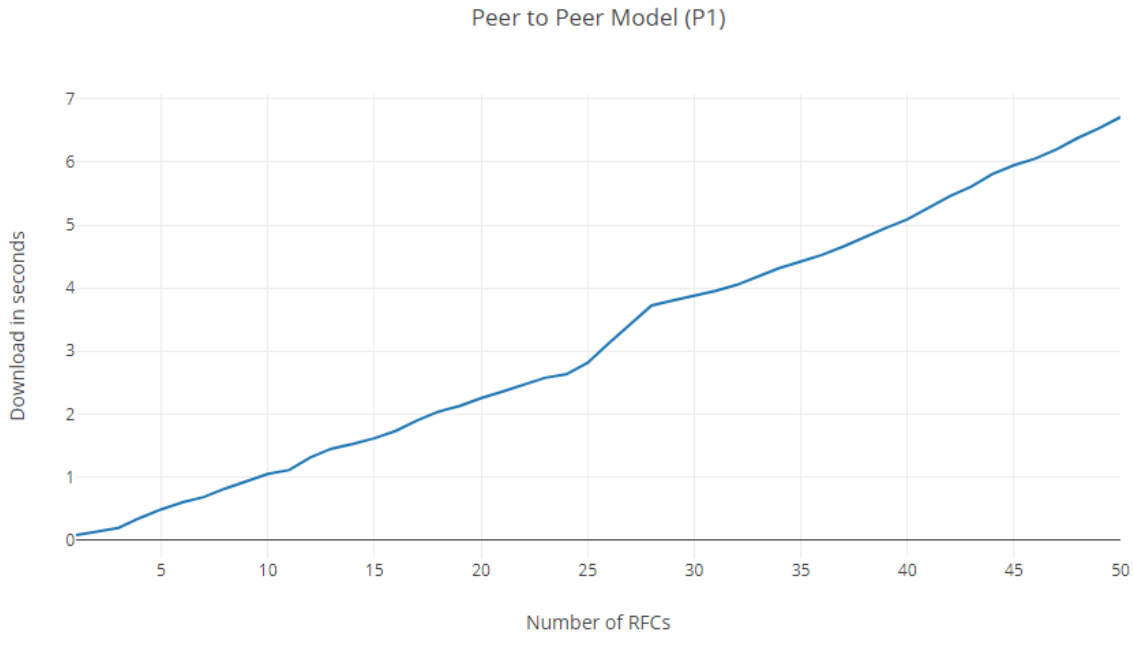

Download in seconds
0
1
2
3
4
5
6
5
10
15
20
25
30
35
40
45
50
Number of RFCs

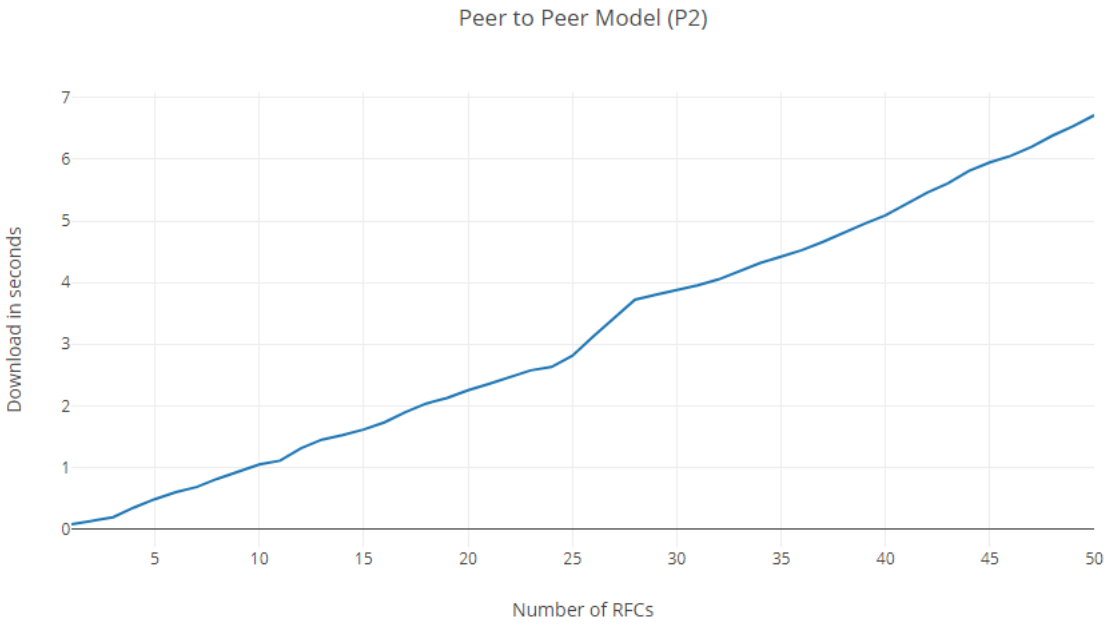

Download in seconds
0
1
2
3
4
5
6
5
10
15
20
25
30
35
40
45
50
Number of RFCs

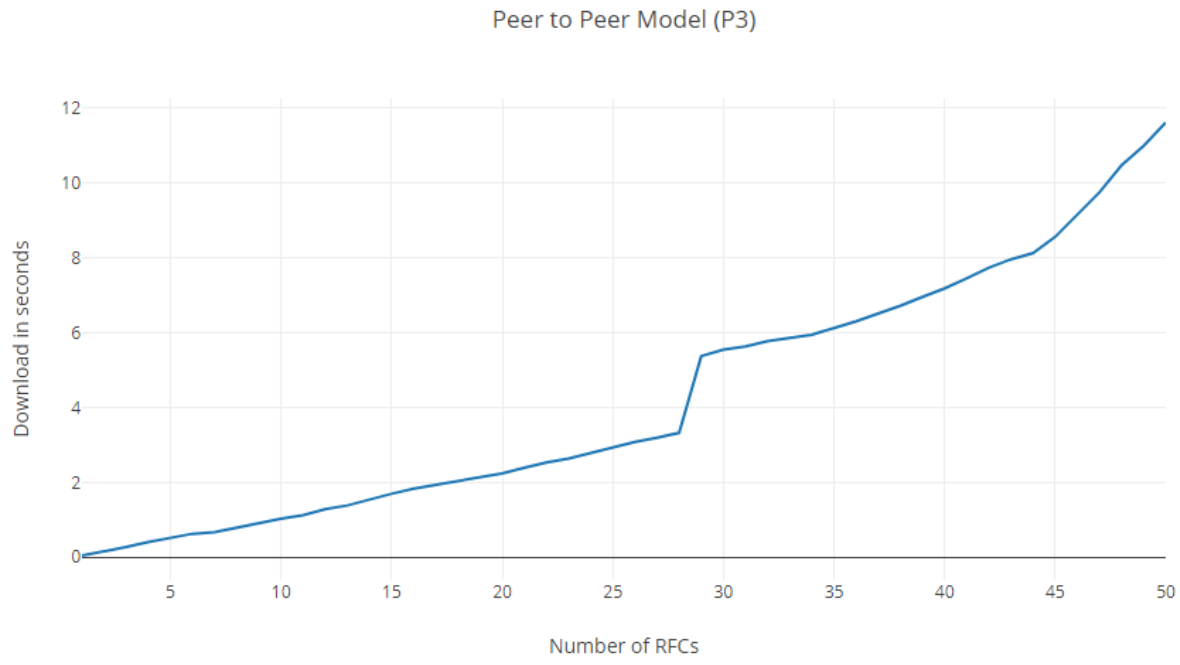


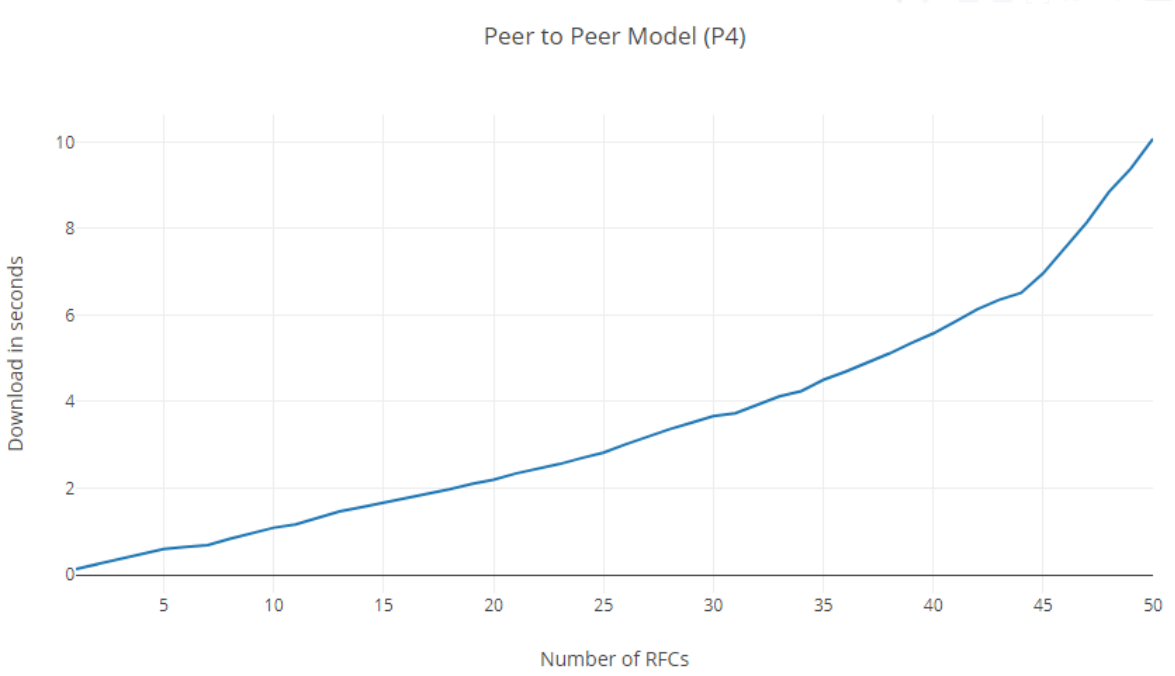
Worst case scenario:

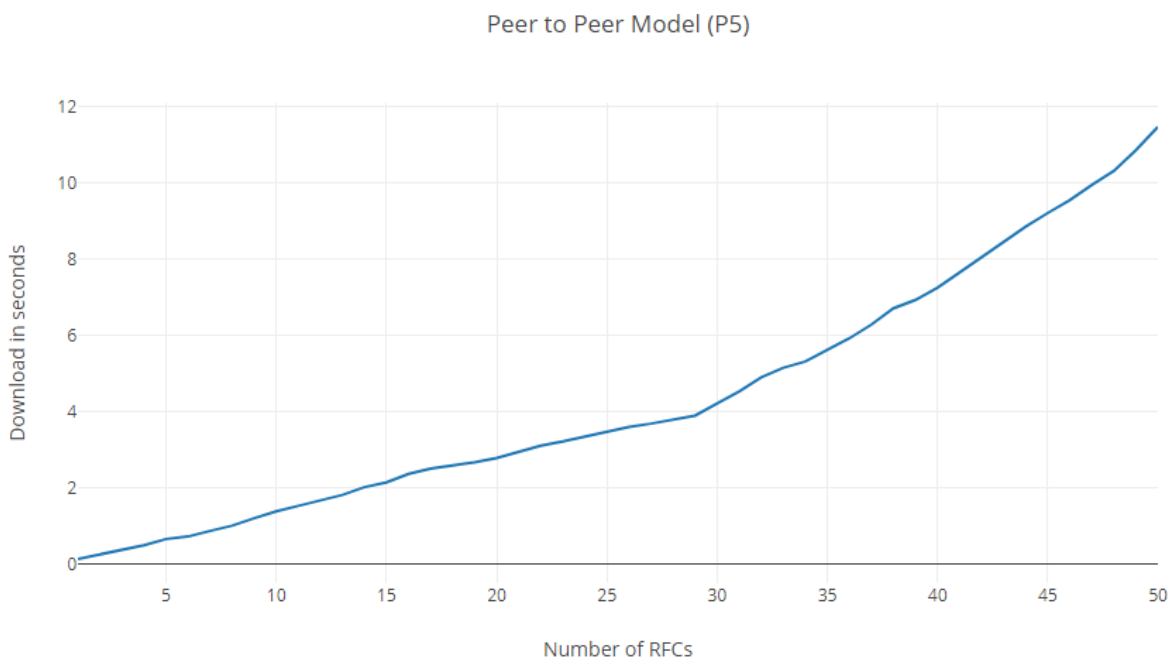





Peer to Peer Model (P2)
Download in seconds
0
1
2
3
4
5
6
7
5
10
15
20
25
30
35
40
45
50
Number of RFCs

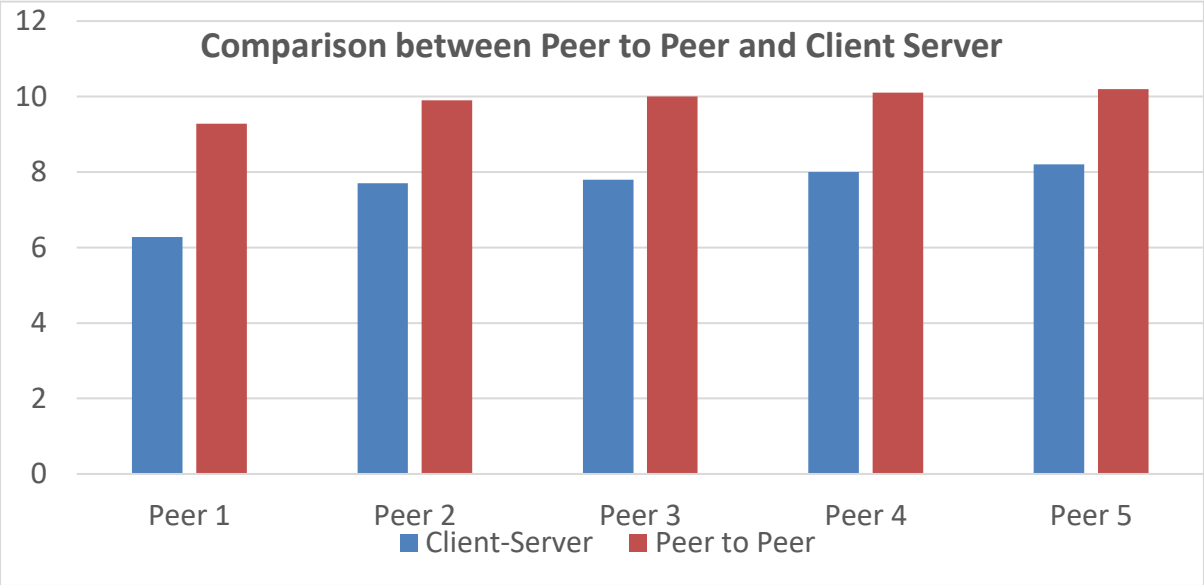

Peer to Peer Model (P3)
Download in seconds
0
2
4
6
8
10
12
5
10
15
20
25
30
35
40
45
50
Number of RFCs


Peer to Peer Model (P4)
Download in seconds
0
2
4
6
8
10
5
10
15
20
25
30
35
40
45
50
Number of RFCs


Peer to Peer Model (P5)
Download in seconds
0
2
4
6
8
10
12
5
10
15
20
25
30
35
40
45
50
Number of RFCs

E. Observations and Conclusion:

Graph of Cumulative distribution time in seconds for peer to peer and client server for each peer**.**



1)As we can see that the download time curve for peer to peer and client server architecture is almost similar with peer to peer system taking slightly more time. This is because each peer is taking some time to send and process GETRFC requests. In all, both the architectures perform equally well for tasks given for 6 peers.

2)But, as we will increase the number of peers, the load on single server will increase as it has to manage all the peers together. This may lead to congestion and load on server. Thus, the client-server will not be scalable for high number of peers.

3)In case of peer to peer, since the database is distributed, even if we increase the number of peers, there won't be much load on the single peer-server as the load will be distributed equally (average case) among all the peers.

Thus, in conclusion, from this project, we found that peer to peer architecture is more scalable as compared to client-server architecture.