**PROJECT REPORT**

1. **Message Formats:**

The following are the message formats used in our Client-Server and Peer-to-Peer communication.

**From Client to RS server**

1. **Register:** REGISTER<sp>hostname\_value<cr>OS:<sp>OSName\_value<cr>ADDRESS<sp>client\_ip\_value<cr>PORT<sp>port\_no\_value<cr>REQ-COOKIE<sp>request\_a\_cookie\_value
2. **Leave** : LEAVE<sp>hostname\_value<cr>OS:<sp>OSName\_value
3. **Keep alive:** KEEPALIVE<sp>hostname\_value<cr>OS:<sp>OSName\_value
4. **Pquery**: PQUERY<sp>hostname\_value<cr>OS:<sp>OSName\_value

**From RS to client**

If client is not registered, then following messages are sent

1. **Register request reply**:

Host hostname\_value registered!

Cookie : cookie\_value

Flag : flag\_value

TTL : ttl\_value

No of times you are active : peer\_time\_active\_value Most recent registration registration\_date\_of\_peer\_value

1. **Leave request reply**

Host hostname\_value Not Registered!

1. **Keep alive reply**

Host hostname\_value Not Registered/ACTIVE!

If client is already registered, then following messages are sent:

1. **Register request reply**

Host hostname\_value already registered, updating values!

Cookie : cookie\_value

Flag : flag\_value

TTL : ttl\_value

No of times you are active : peer\_time\_active\_value Most recent registration registration\_date\_of\_peer\_value

1. **Leave message reply**

Host hostname\_value leaving the system

Cookie : cookie\_value

Flag : flag\_value

TTL : ttl\_value

No of times you are active : peer\_time\_active\_value Most recent registration registration\_date\_of\_peer\_value

1. **Keep alive message reply**

Keep Alive Message Received from hostname\_value

Host hostname\_value Registered again using KeepAlive!

1. **PQuery message reply**

If no peers are active, it returns

null

Else it returns

hostname\_value<sp>client\_ip\_value<sp>port\_no\_of\_RFC\_server\_value<cr>

**From Peer Client to Peer Server**

1. **RFC Query message:**

GET<sp>RFC-INDEX<sp>P2P-DI/1.1<cr><tr>HOST : <sp>hostname\_value<cr><tr>OS : <sp>OSName\_value<cr><tr>

1. **Get RFC message:**

GET<sp>RFC<sp>RFC\_number\_value<sp>P2P-DI/1.1<cr><tr>HOST : <sp>hostname\_value<cr><tr>OS : <sp>OSName\_value<cr><tr>

**From Peer Server to Peer Client**

1. **RFC query Response message:**

P2P-DI/1.1<sp>200<sp>OK<cr><tr>RFCResponse<cr><tr> RFCNO<sp> RFC\_number\_value<cr>RFCNAME<sp>RFC\_name\_value<cr>HOSTNAME<sp> hostname\_value<cr>IP<sp> client\_ip\_value<cr><tr>

1. **TASKS**

**TASK - 1**

**TASK – 2 (BEST CASE)**

**TASK – 2 (WORST CASE)**

**Conclusions**

From the graphs of Task-1 and Task-2, we see that in Task-1 the graph increases linearly because all the clients tries to download the RFC’s from the server linearly. In this case, the server accepts one request at a time, process it, and then moves on to service another client.

Whereas, in the best case scenario of Task-2, where all the clients interacts with each other, the graph increases linearly for first few RFC’s. As the time passes, and each client gets RFC’s, the graph flattens out a little indicating several peers are downloading files from multiple clients. Hence, the download time in the end becomes constant for each peer instead of increasing gradually thus showing the advantage of peer2peer download.

In the worst case scenario of Task-2, where one client first asks for all the files from peers and then next peer starts downloading files from other peers and so on, the graph tends to become similar to Task-1 graph.

From the above graphs, it is concluded that for file transfer systems, P2P systems are more scalable than the centralized systems as P2P systems will download the files more quickly than client-server systems as each client in P2P system add its own capacity to the system. Also, it puts less load in terms of file transfer on the file server.