Project Report

For our protocol, we have defined a generic format of Request and response message.

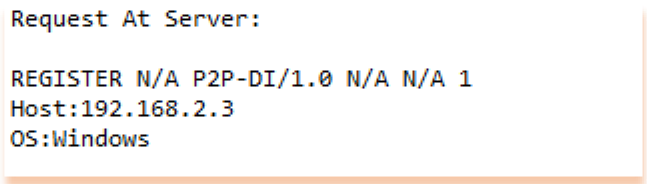
Message Format :
method <sp> document <sp> version <sp> Status code <sp> Phrase <sp> cookie <cr> <lf>
header field name <sp> value <cr> <lf>
header field name <sp> value <cr> <lf>
<cr> <lf>

If some field is not present in the request, we have displayed it as N/A at the console.
Also, following are the status code and phrases we have defined in the protocol.

101	OK
102	Already Registered
110	Not Authorized
306	Success

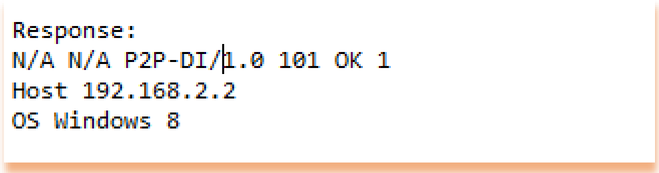
1. **Peer to Registration Server Communication :**

a. **Register:**



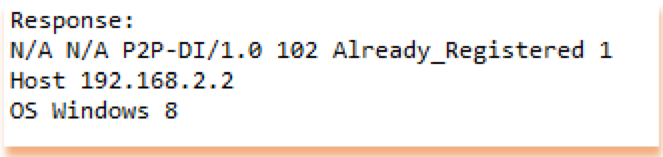
```
Request At Server:

REGISTER N/A P2P-DI/1.0 N/A N/A 1
Host:192.168.2.3
OS:Windows
```

First Time Registration:



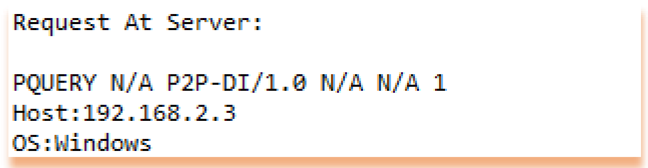
```
Response:
N/A N/A P2P-DI/1.0 101 OK 1
Host 192.168.2.2
OS Windows 8
```

On Trying to register again:

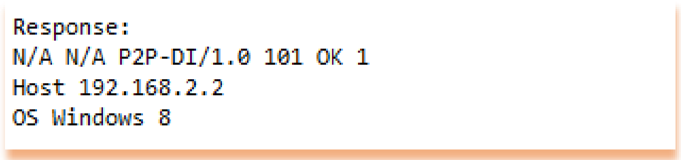


```
Response:
N/A N/A P2P-DI/1.0 102 Already_Registered 1
Host 192.168.2.2
OS Windows 8
```

b. PQuery

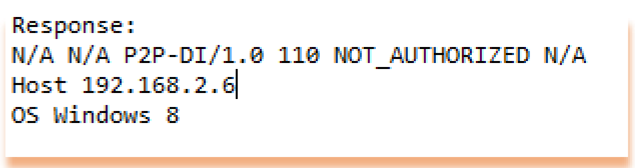


```
Request At Server:

PQUERY N/A P2P-DI/1.0 N/A N/A 1
Host:192.168.2.3
OS:Windows
```



```
Response:
N/A N/A P2P-DI/1.0 101 OK 1
Host 192.168.2.2
OS Windows 8
```

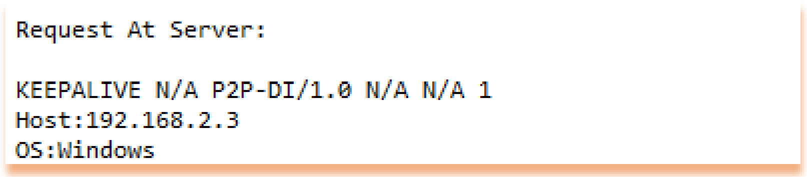
PQuerying and If Not Registered



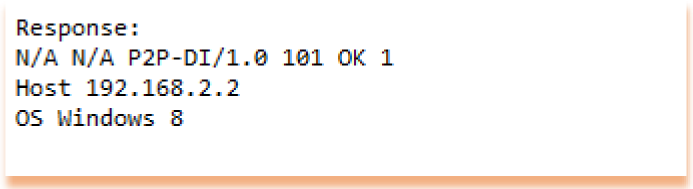
```
Response:
N/A N/A P2P-DI/1.0 110 NOT_AUTHORIZED N/A
Host 192.168.2.6
OS Windows 8
```

List Of Active Peers is sent as a Serialized Object to the Peer which is deserialized at the peer side to get information of active Peers.

c. Keep Alive

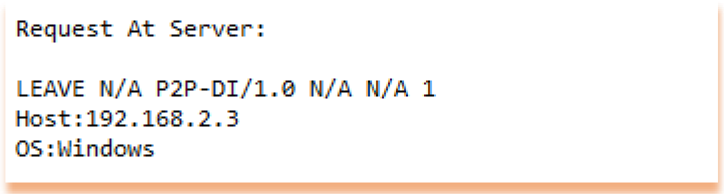


```
Request At Server:

KEEPALIVE N/A P2P-DI/1.0 N/A N/A 1
Host:192.168.2.3
OS:Windows
```


```
Response:
N/A N/A P2P-DI/1.0 101 OK 1
Host 192.168.2.2
OS Windows 8
```

Here Keep Alive resets the TTL value of peer to 7200 seconds at the server.

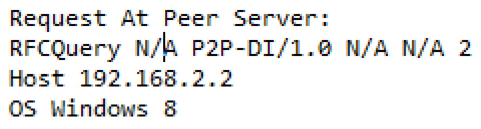
d. Leave

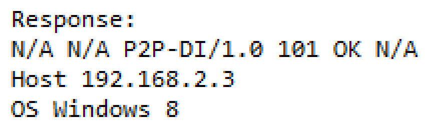

```
Request At Server:

LEAVE N/A P2P-DI/1.0 N/A N/A 1
Host:192.168.2.3
OS:Windows
```

No response from Server as the Peer status is set to inactive and it is no more in the network.

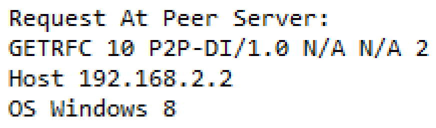
1. Peer to Peer Communication :

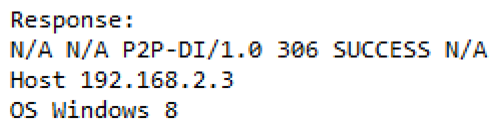
a. RFC Query :


```
Request At Peer Server:
RFCQuery N/A P2P-DI/1.0 N/A N/A 2
Host 192.168.2.2
OS Windows 8
```


```
Response:
N/A N/A P2P-DI/1.0 101 OK N/A
Host 192.168.2.3
OS Windows 8
```

b. GETRFC :

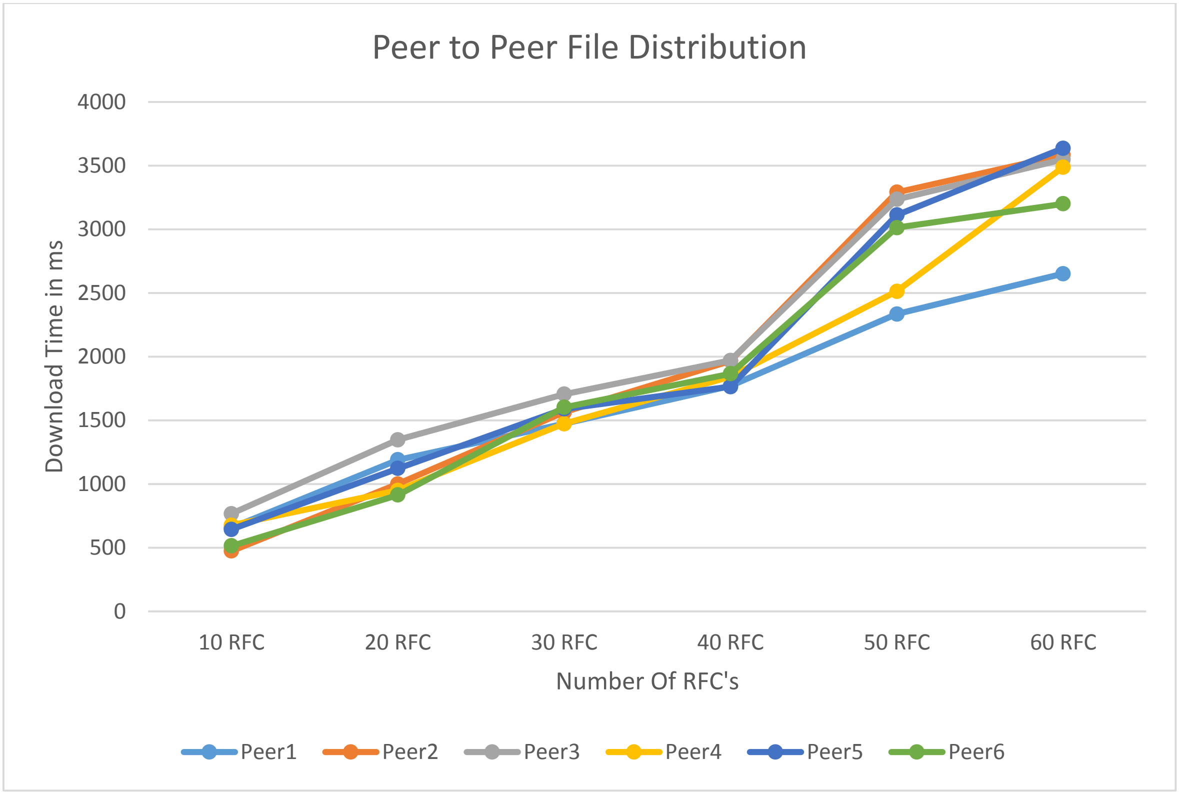

```
Request At Peer Server:
GETRFC 10 P2P-DI/1.0 N/A N/A 2
Host 192.168.2.2
OS Windows 8
```


```
Response:
N/A N/A P2P-DI/1.0 306 SUCCESS N/A
Host 192.168.2.3
OS Windows 8
```

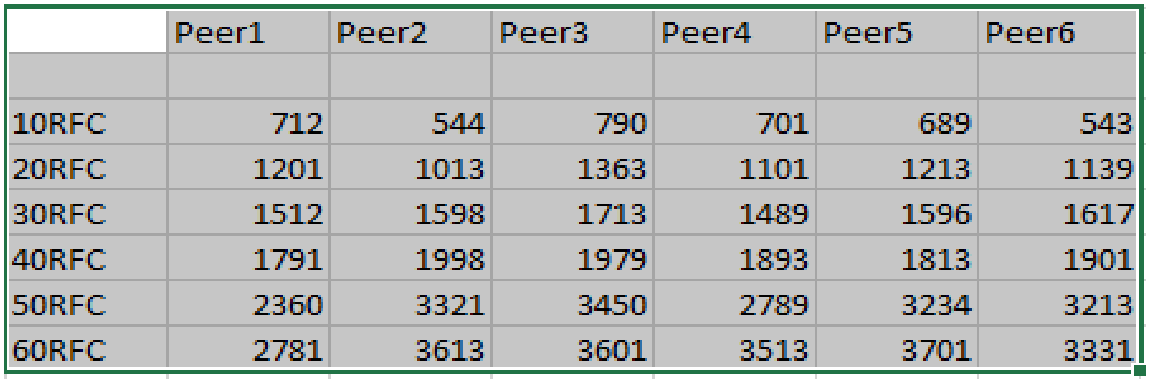
2. Peer to Peer Communication Download Curves :

Best Case (Multithreaded Peer): Here Peers run multiple thread in parallel to request for RFC's which leads to multiprocessing and reducing the time**.**

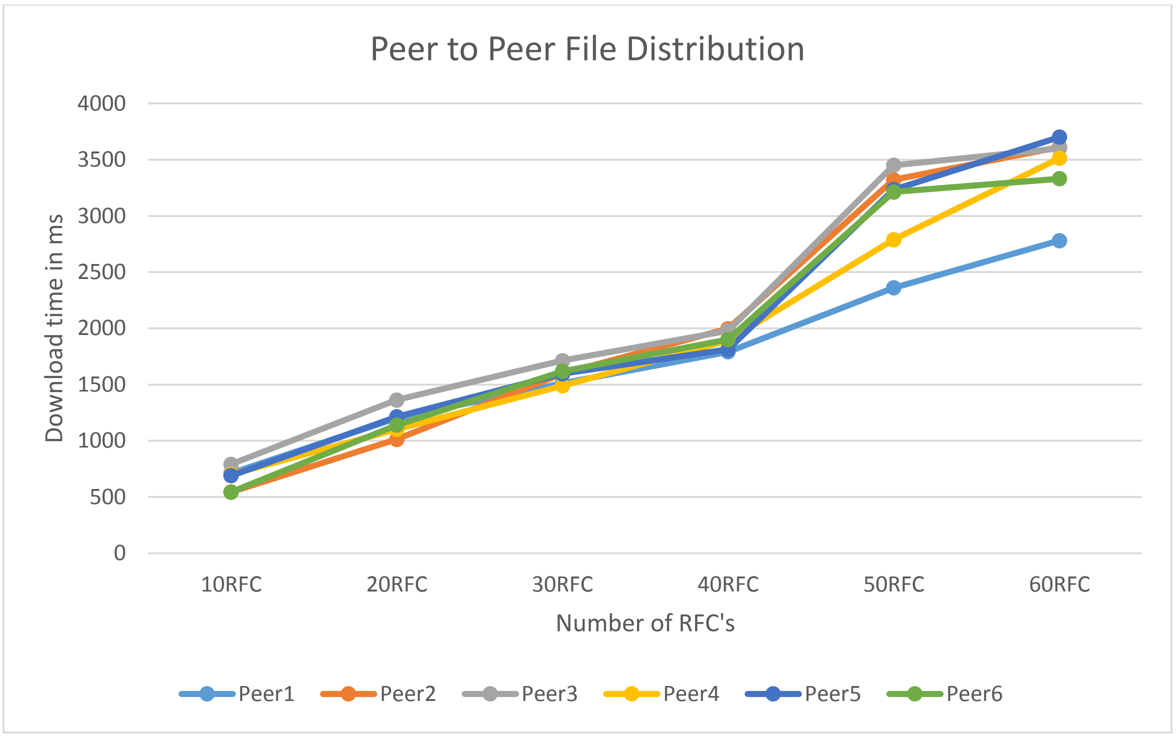
	Peer1	Peer2	Peer3	Peer4	Peer5	Peer6
10 RFC	655	473	766	675	643	513
20 RFC	1189	999	1346	949	1122	913
30 RFC	1473	1562	1705	1472	1589	1601
40 RFC	1770	1966	1970	1842	1764	1865
50 RFC	2334	3290	3235	2512	3112	3012
60 RFC	2650	3583	3548	3487	3635	3200



Worst Case (Single Threaded Peers): Peers are single-threaded and will download each RFC sequentially here.

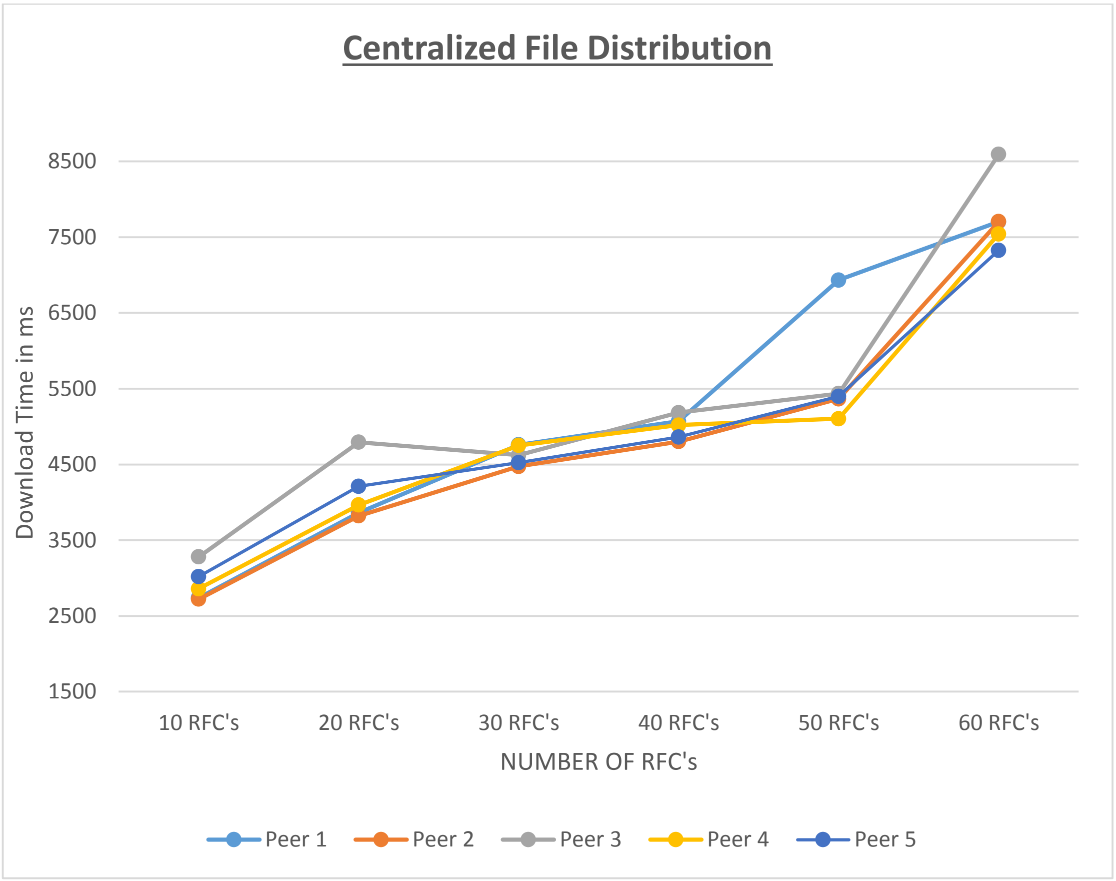


	Peer1	Peer2	Peer3	Peer4	Peer5	Peer6
10RFC	712	544	790	701	689	543
20RFC	1201	1013	1363	1101	1213	1139
30RFC	1512	1598	1713	1489	1596	1617
40RFC	1791	1998	1979	1893	1813	1901
50RFC	2360	3321	3450	2789	3234	3213
60RFC	2781	3613	3601	3513	3701	3331



3. Centralized File Distribution System Download Curves :

	Peer 1	Peer 2	Peer 3	Peer 4	Peer 5
10 RFC's	2738	2723	3279	2859	3021
20 RFC's	3863	3822	4794	3965	4213
30 RFC's	4760	4473	4621	4750	4524
40 RFC's	5069	4802	5184	5019	4862
50 RFC's	6935	5365	5435	5103	5398
60 RFC's	7703	7705	8593	7542	7325



4. From the download curves, it can be seen that for Centralized file distribution, the time taken to download the RFC’s is more as compared to that of Peer to Peer File distribution. It can be seen that the Peer 2 peer distribution is almost twice as efficient (For 60 RFC’s) as Centralized file distribution.

There are two different cases in Peer to Peer distribution as well. First case is when the system is multi-threaded and the file downloading takes place in parallel. This means that the client will spawn one thread to download one RFC which will then execute till the RFC is downloaded. This makes sure that the download times of RFC’s overlaps and we get better download times.

In the second case, the system is single-threaded and the files are downloaded in the sequential manner. The client process will call the function that will create a connection with the server holding the file. It will then download that file and once it is done, it will close the connection. Again, for second file, it will open a new connection and downloads it. This will take more time as the download times do not overlaps with each other.

Even though the time difference between both the case looks small in our results, this time difference will increase linearly as the number of RFC’s to be downloaded increases. In cases, where huge number of files need to be downloaded, the multi-threaded system will work much better than single threaded system and will deliver better download times.

Since in the current report, we are presenting results for only 60 RFC’s and 6 peers that are downloaded. Since, these requests are within the computing power of the CPU, the centralized system does perform well as compared to when millions of requests are received for downloading. If such a high number of requests are received by the single peer, the peer can only process a limited amount of them at a time and thus need to buffer the remaining requests. This is the reason why Google maintain a large number of servers in distributed manner. If the number of internet users increases to double, then Google will have to deploy more servers to process the increased requests. So, the Centralized download curve increases linearly.

One the other hand, Peer 2 Peer systems are quite scalable as a new peer can enter the system and can share the resources it has with others and can use others resources as well. So, as the load will increases, the