Michael Patel
Venkata Surya Teja Penumatcha
ECE 573 Project 1 Report

Overview

The peer-to-peer (P2P) network system operates with a distributed index (DI) architecture. As per figure 1, a registration server RS is always on and interacts with many peers directly. The RS has a server port number that is known in advance to all peers. The purpose of the RS is to keep information on active peers in the system. The RS does not keep content information about what each peer has on its own RFC servers. Instead, it facilitates information about the system as a whole. The RS uses a dictionary data structure Peer Index to maintain information about the peers.

Each peer has both its own RFC client and RFC server. This allows for peers to communicate directly with each other when exchanging the content data. Each peer's RFC server maintains a list data structure RFC index which details what specific RFC documents the local peer has. The peer's RFC server handles the data exchange with a contacting peer.

The communication between a peer and the RS can take place through four different message exchanges (registration, leave, keep-alive and peer-query). The registration message adds the contacting peer to the RS peer index. This should be the first message a peer sends to the RS. The leave message deletes/deactivates the contacting peer from the RS peer index. The keep-alive message contacts the RS to let it know that it is still live in the system. The peer-query message returns a list of active peers as a dictionary data structure to the contacting peer. All communication is initiated by the peer RFC client.

The communication between peers involves actual data exchange. There are two types of message exchanges (rfc-query and get-rfc). The rfc-query message returns a peer's local RFC index. This message should be sent before requesting to download RFC documents. The get-rfc message performs the actual RFC document exchange between peers.

The individual and total download times are recorded in separate text files and a plot with the curves can be generated using "python plot.py"
All communication between the peers and RS takes place over TCP. Each message handles its own TCP connection independently.

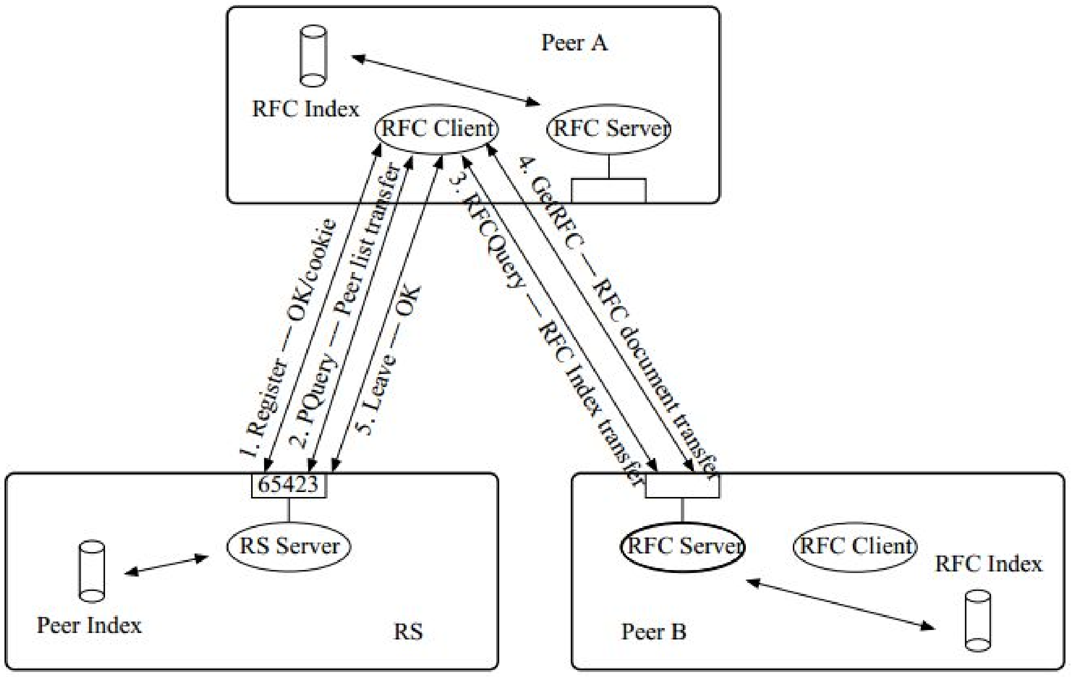
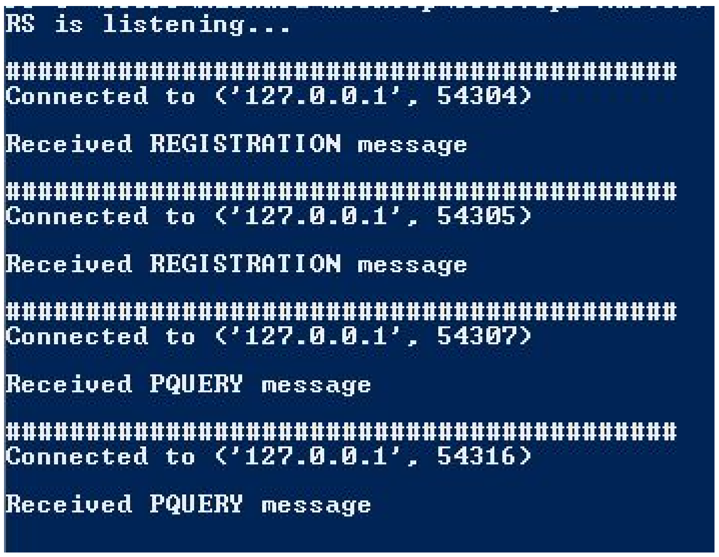


Figure 1: Sequence of messages in the P2P-DI system

Format of messages exchanged:

Peer-to-RS Communication format:

```
############################################
Connected to < IP ADDRESS, PORT NUMBER > < cr > < lf >
< lf >
Received < MESSAGE > message < cr > < lf >
< lf >
```

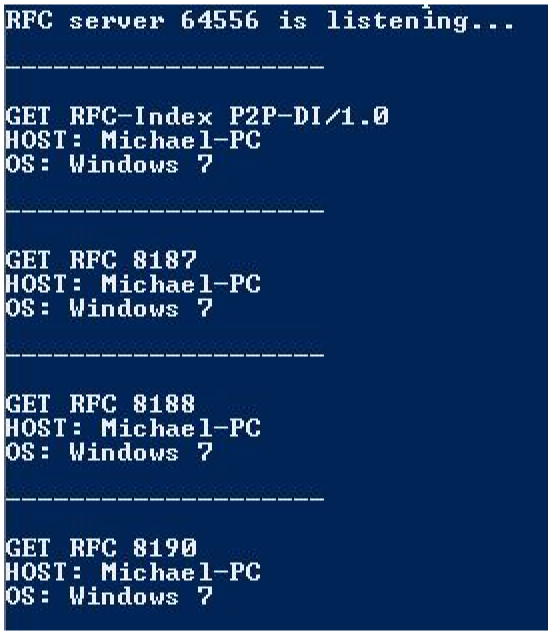


```
RS is listening...

############################################
Connected to ('127.0.0.1', 54304)

Received REGISTRATION message

############################################
Connected to ('127.0.0.1', 54305)

Received REGISTRATION message

############################################
Connected to ('127.0.0.1', 54307)

Received PQUERY message

############################################
Connected to ('127.0.0.1', 54316)

Received PQUERY message
```

Figure 2: RS serving peer clients' requests for registration and peer-query

Peer-to-Peer Communication format:

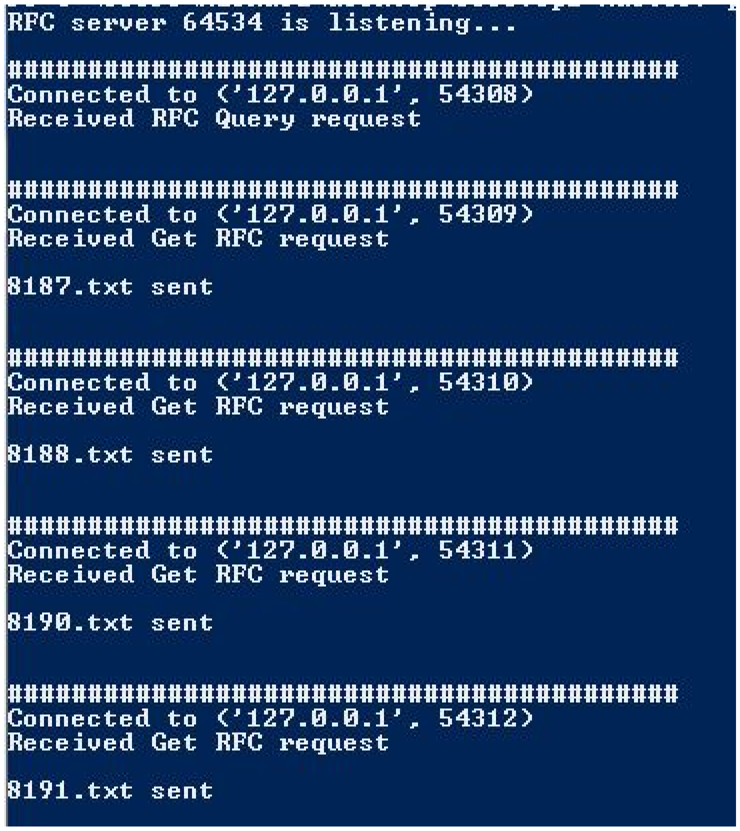
GET < RFC REQUEST TYPE > < cr > < lf >
HOST < HOSTNAME > < cr > < lf >
OS < OPERATING SYSTEM TYPE > < cr > < lf >
< lf >



```
RFC server 64556 is listening...

--------------------

GET RFC-Index P2P-DI/1.0
HOST: Michael-PC
OS: Windows 7

--------------------

GET RFC 8187
HOST: Michael-PC
OS: Windows 7

--------------------

GET RFC 8188
HOST: Michael-PC
OS: Windows 7

--------------------

GET RFC 8190
HOST: Michael-PC
OS: Windows 7
```

Figure 3: Peer client requesting another peer's RFC index and RFC documents

##
Connected to < IP ADDRESS, PORT NUMBER > < cr > < lf >
Received < RFC REQUEST MESSAGE > < cr > < lf >
< lf >
< RFC DOCUMENT > sent < cr > < lf >
< lf >



```
RFC server 64534 is listening...

##########################################
Connected to ('127.0.0.1', 54308)
Received RFC Query request


##########################################
Connected to ('127.0.0.1', 54309)
Received Get RFC request

8187.txt sent


##########################################
Connected to ('127.0.0.1', 54310)
Received Get RFC request

8188.txt sent


##########################################
Connected to ('127.0.0.1', 54311)
Received Get RFC request

8190.txt sent


##########################################
Connected to ('127.0.0.1', 54312)
Received Get RFC request

8191.txt sent
```

Figure 4: Peer server serving another peer’s requests for its RFC index and RFC documents

Download Time Curves:

Task 1: Centralized File Distribution

Six peers are created. One peer has all sixty RFC documents. The other five peers do not have any RFC documents initially. They all first register with the RS. Then, each peer requests the active peer list from the RS. The five peers who do not have any RFC documents contact the peer who has all of them separately and begin downloading from this "central" peer server. The five peers download a total of fifty RFC documents each from the central peer server. The plot below illustrates the download times for this system configuration.

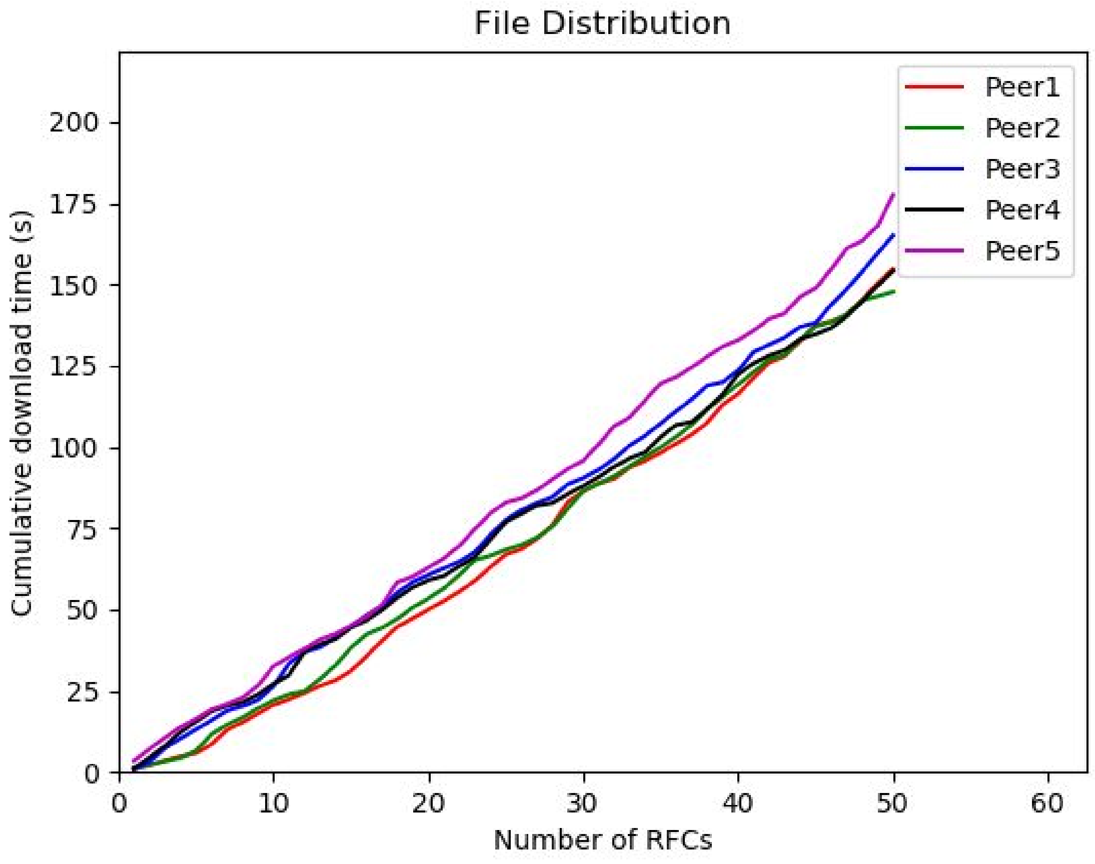


Figure 5: Task 1 Centralized File Distribution Download Curve - Windows 7

Task 2: P2P File Distribution

Six peers are created. Initially, each peer has ten unique RFC documents. They all first register with the RS. Then, each peer requests the active peer list from the RS. Each peer contacts other peers in the system to request and download RFC documents until each peer has all sixty RFC documents. The plots below illustrate the download times for this system configuration.

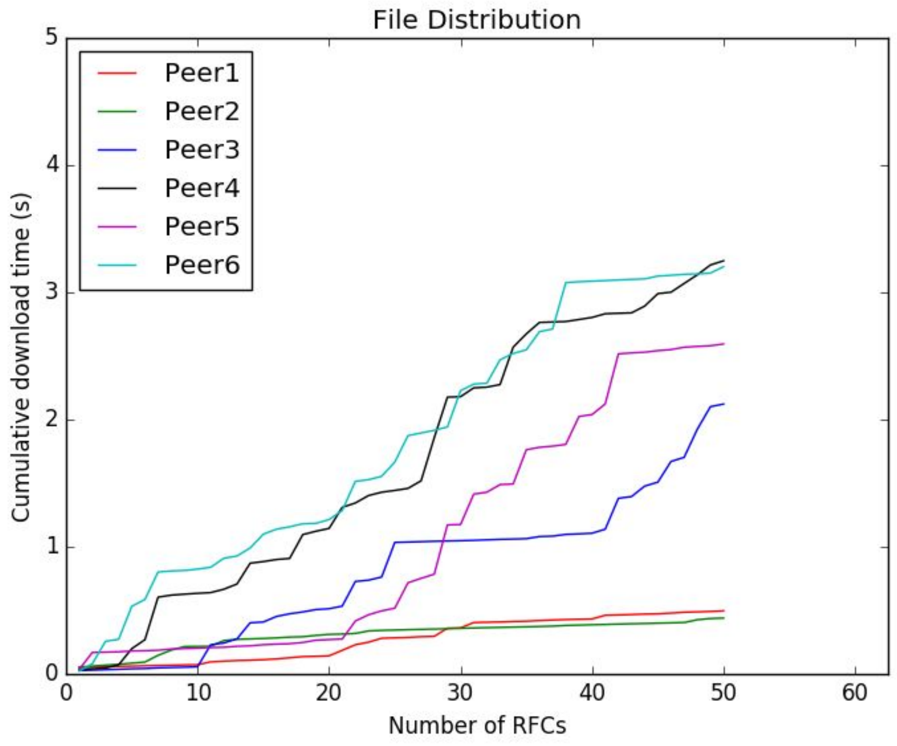


Figure 6: Task 2: P2P File Distribution (best-case) Download Curve - Mac

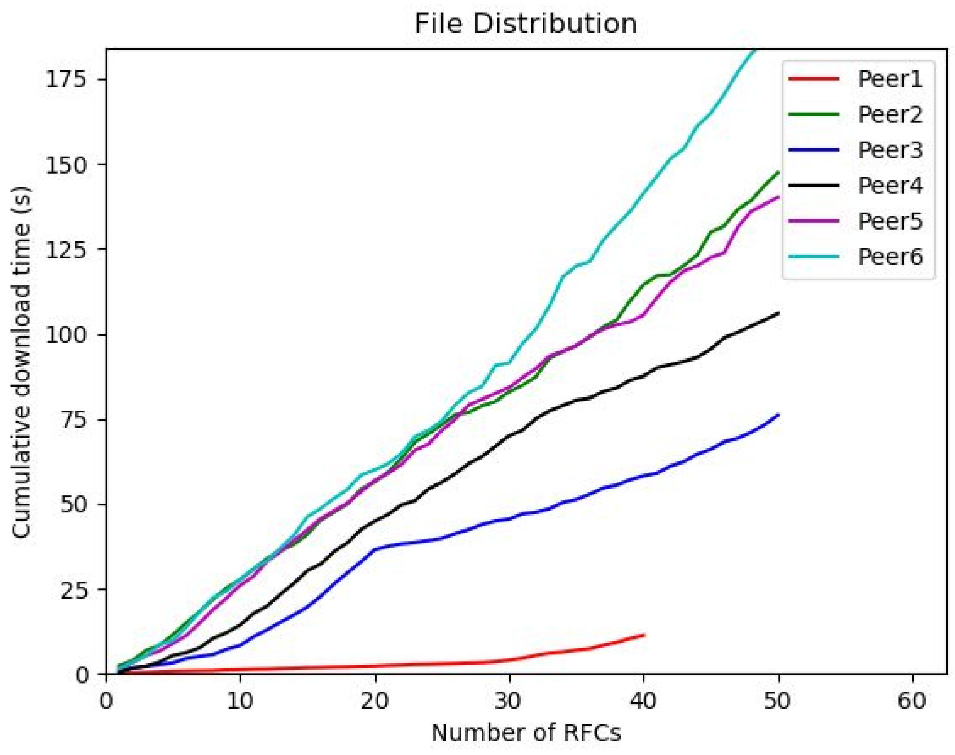


Figure 7: Task 2: P2P File Distribution (best-case) Download Curve - Windows 7

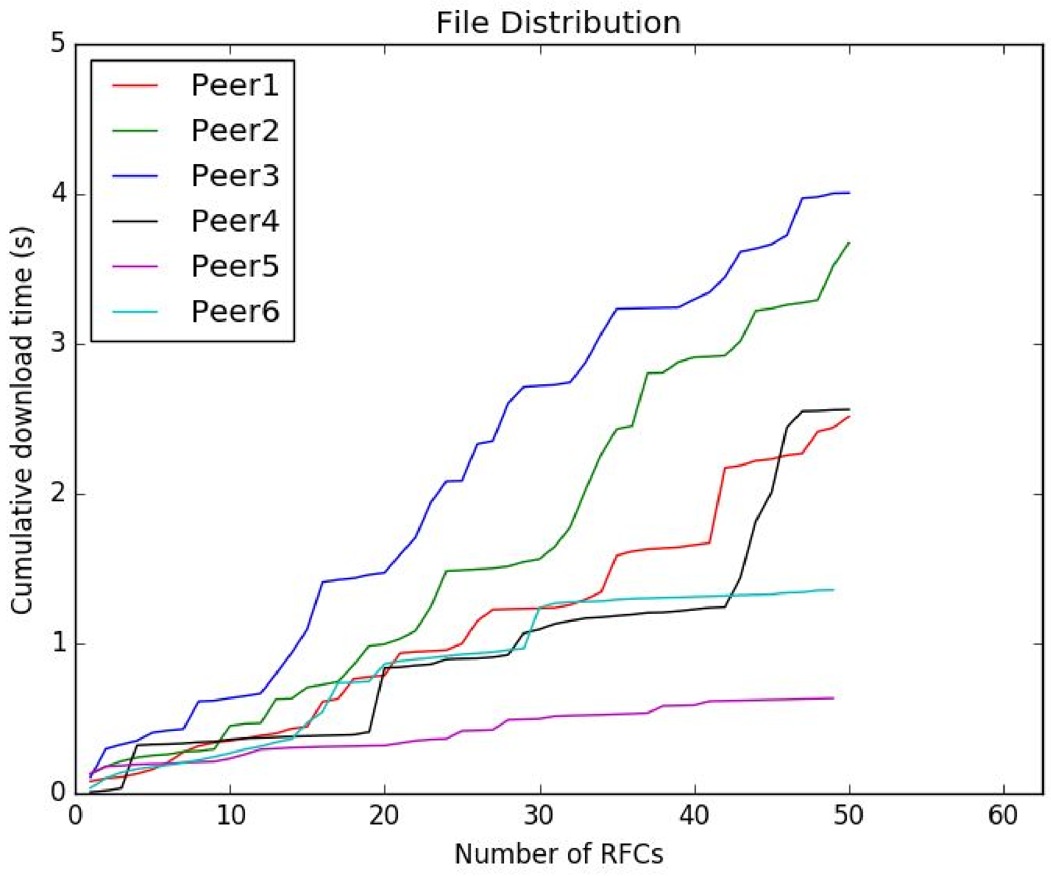


Figure 8: Task 2: P2P File Distribution (worst-case) Download Curve - Mac

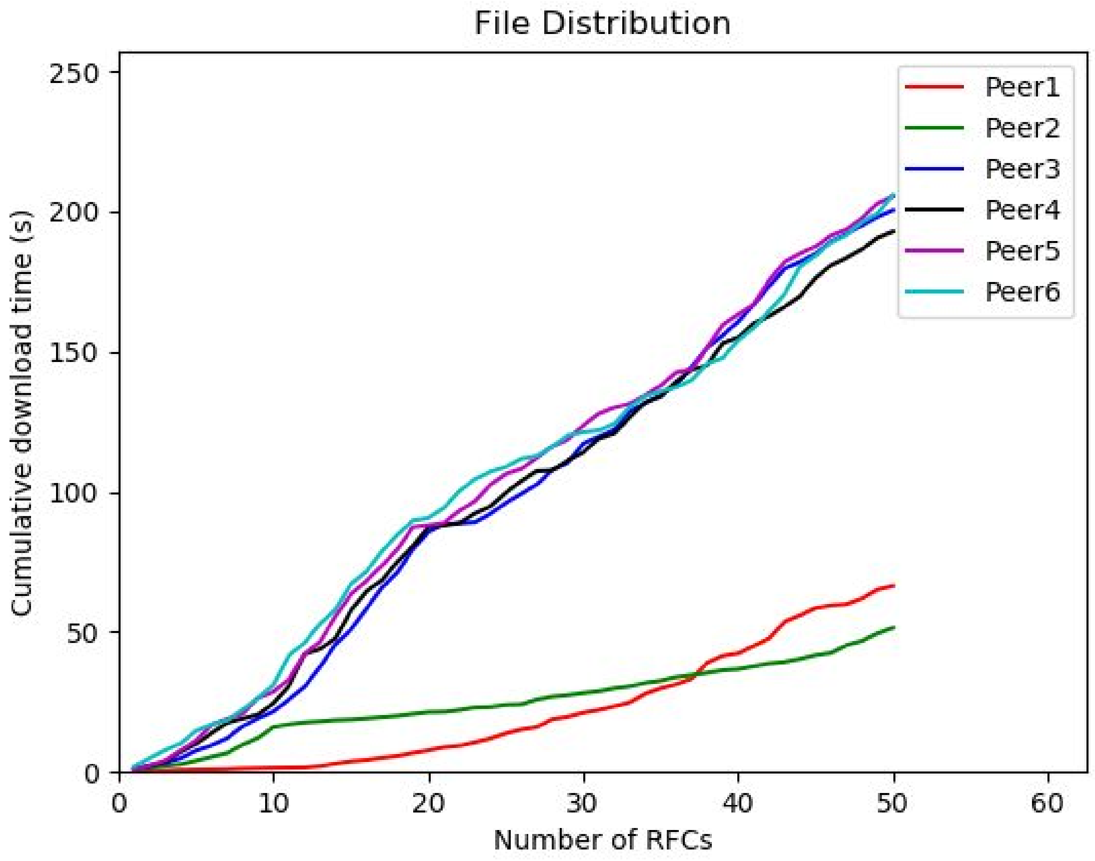


Figure 9: Task 2: P2P File Distribution (worst-case) Download Curve - Windows 7

Conclusion:

In task 1, we observe that in the centralized file distribution, the download curve is linear. All the peers are downloading files at about the same rate. The cumulative download time is quite long.

In task 2 (Peer-to-Peer distribution) the first peer is downloading the files faster and the latest-to-join peer is receiving the files at a slower rate. In the worst case scenario, we notice that all the peers start receiving at the same rate, but when some peers start to leave (i.e. they are selfish) the transfer rate decreases for the other peers. This can be observed from the increase in slope for some peers in Figures 8 and 9 above. In the worst case, the peers' trend lines start to deviate from best-case P2P towards centralized distribution. This makes sense as peers leave the system, so does their ability to upload. Thus, the peer-client to peer-server ratio skews towards a situation that looks more like one server serving many peers, rather than many servers serving many clients. However, in the best case scenario, peers are altruistic towards other peers and everyone stays in the system to serve RFC requests. This ultimately leads to a lower cumulative download time for all peers. In both P2P configurations, the cumulative download times were still less than in the client-server architecture. Therefore, P2P is a better system when cumulative download time is the only goal.

Additionally, the overall cumulative download times were much faster on the Mac with Python 2.7 than on the Windows 7 machine with Python 3.6. The time factor was very significant (5s compared to 200s).