
PEER TO PEER FILE SHARING

Computer Systems II – PROJECT 2

OBJECTIVE:

1. A brief survey summarizing existing P2P architectures for file sharing.
2. To develop a simple peer-to-peer (P2P) file sharing system that uses a centralized directory server.
3. A demonstration of the developed P2P system using at least three different hosts.

GROUP MEMBERS

1. Deepthy Mariyam George
ASU ID: 1208621176
Email: dmgeorg3@asu.edu
2. Lakshmi Srinivas
ASU ID: 1208635554
Email: lsriniv2@asu.edu
3. Navneet Jha
ASU ID: 1207987452
Email: njha4@asu.edu

COURSE :

CEN 502 - Computer Systems II - Networks

INSTRUCTOR:

Dr. Violet R. Syrotiuk

SEMESTER

Fall 2015

CONTENTS

List Of Figures	2
1 Introduction	3
2 Classification of P2P File Sharing System	3
2.1 Degree of Centralization	3
2.1.1 Purely Decentralized	3
2.1.2 Partially Centralized	4
2.1.3 Hybrid Decentralized	4
2.2 Network Structure	4
2.2.1 Unstructured P2P Networks	5
2.2.2 Structured P2P Networks	5
2.2.3 Loosely Structured P2P Networks	5
3 Study on Existing File sharing Application	5
3.1 Freenet	5
Communication protocol	5
3.2 Gnutella	6
Communication protocol	6
3.3 CHORD	6
Protocol [4]	7
4 References	8
1 Design Methodology	9
1.1 Central Directory	9
1.1.1 Working and Functionality	9
1.1.2 Message format	9
1.2 Peer	10
1.2.1 Working and Functionality	10
1.2.2 Message format	10
2 Pseudo Code	10
2.1 Central Directory	10
2.2 Peer	11
3 Test and Sample Output	12
4 Conclusion	16
5 References	16

LIST OF FIGURES

Figure 1: Purely Decentralized Architecture	4
Figure 2: Server Meditated P2P	4
Figure 3: Logical flow of Freenet Messages	6
Figure 4: Logical flow of Messages in a Gnutella Network	6
Figure 5:Chord node Identifier Circle Figure 6: Chord Circle with Finger Table	7
Figure 7: Message format for Peer to Central Directory	9

Part I: Survey of Peer to Peer Architectures

1 INTRODUCTION

P2P computing is defined as the sharing of computer resources and services by direct exchange between systems. The resources and services include the exchange of information and content files, processing cycles, cache storage, and disk storage for data files. P2P takes advantage of existing desktop computing power and networking connectivity allowing economic clients to leverage their collective powers and benefit the entire community. Each of the workstation on the network can act as a client or server. These systems try to address and overcome the limitations of the client-server architectures. Peer to peer networks are characterized by peer autonomy, dynamic nature of connectivity i.e., peers can leave the network or get added to it dynamically and direct collaboration with other peers.

File sharing is the most dominant applications on the internet that uses a Peer-to-peer network system. The P2P client will always initiate the communication request. P2P file sharing architectures can be classified on their degree of centralization, i.e., the extent to which they rely on one or more servers to facilitate their interaction with another peer. Highly dynamic p2p networks of peers with complex topology can be differentiated by the degree to which they contain some structure or are created adhoc.

2 CLASSIFICATION OF P2P FILE SHARING SYSTEM

As mentioned above, the P2P file sharing systems can be broadly classified based on their degree of centralization or their network structure and are discussed in the sections to follow.

2.1 DEGREE OF CENTRALIZATION

This classification is based on their dependence on servers for the purpose of facilitating the communication as Purely decentralized, Partially centralized and Hybrid decentralized.

2.1.1 Purely Decentralized

All the machines in the network framework perform the same set of task, can act as both the server and the client. These are often referred to as the 'Pure P2P' systems. All nodes in the network perform exactly the same tasks, acting both as servers and clients, and there is no central coordination of their activities. The nodes of such networks are termed "servents" (SERVers+clieENTS). The strength of this type of system is that it doesn't rely on any central servers in order for peers to locate and connect to it. This at the same time poses a problem that lesser number of clients/peers can be found. This problem can be overcome using network broadcasting and discovery techniques, like IP Multicast, though that has its own limitations since it isn't widely deployed to the internet and invites a denial of service attack. . Pure P2P is also deployed to the Internet in the case that non-multicast schemes are used to discover peers s. In this case, the application may use some other schemes for peer discovery such as a well-known approach that each peer knows about at least one other peer and they share this knowledge with other peers to form a loosely connected mass of peers

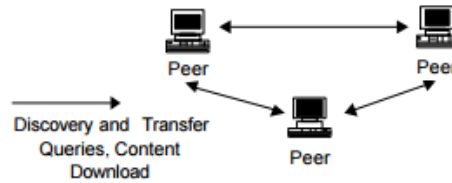


Figure 1: Purely Decentralized Architecture

2.1.2 Partially Centralized

There are two types of partially centralized architectures. The first ones are similar to centralized architectures, but instead of a single server there is a farm of servers with a P2P network at this level. The second ones are similar to decentralized architectures, but there are some nodes called supernodes that act as a central node. This supernodes will perform the search for other supernodes in order to find the requested file. They are dynamically assigned and in case they are subjected to any attack, the network will take action to replace them.

2.1.3 Hybrid Decentralized

There is a central server facilitating the interaction among the various host machines. It maintains a directory of all the shared files on the registered users PC, in the form of metadata. These server-like system only facilitate the peer-to-peer interaction between end systems by performing lookups and identifying the nodes in the network where the files would be found. This type of architecture is often referred to as a Server-mediated architectures. However, this approach is dependent on the availability of the central server. If the central server is not available, the P2P application will not be able to find other peers.

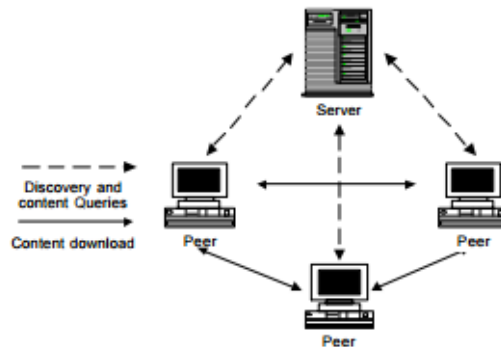


Figure 2: Server Meditated P2P

2.2 NETWORK STRUCTURE

P2p networks consist of highly complex topology since it is a dynamic network of peers. P2P systems can be differentiated by the degree to which these overlay networks contain some structure or are created ad-hoc. By structure here we refer to the way in which the content of the network is located with respect to the network topology. They are classified according to this characteristic as Unstructured, Structured and Loosely structured

2.2.1 Unstructured P2P Networks

The placement of data in such networks is completely unrelated to overlay topology. There is no information regarding which nodes are likely to have relevant files. This makes searching for files tedious, random nodes are probed and asked if they have the file that matches the one that is queried. Unstructured p2p systems differ in the way in which they construct the overlay topology, and the way in which they distribute queries from node to node. The advantage of such systems is that they can easily accommodate a highly transient node population. The disadvantage is that it is hard to find the desired files without distributing queries widely. For this reason unstructured p2p systems are considered to be unscalable

2.2.2 Structured P2P Networks

These type of networks attempt to over the scalability issues related to the unstructured P2P Systems. They replace the random searches for a query by providing a mapping between the file descriptor or identifier and location in a routing table format and present a scalable solution for exact match queries. The disadvantage of deploying such systems is maintenance- often hard to maintain the routing structure in a highly dynamic system where peers join and leave the network frequently.

2.2.3 Loosely Structured P2P Networks

These are architectures that fall in between the above two mention categories. File locations are affected by routing hints, but aren't completely specifies

3 STUDY ON EXISTING FILE SHARING APPLICATION

For the purpose of this survey, three common existing file sharing applications were chosen and their communication protocol was studied. A summary is presented in the following sections.

3.1 FREENET

Freenet is a purely decentralized, loosely structured system. It operates as an adaptive peer to peer network that queries one another to store and receive files. The files will be named by location independent keys. In Freenet each of the nodes maintain their own local datastores and are made available to the network (so that other peers can read and write based on the contents of theses datastores). They have a dynamic routing table with address information of the other peers in the network and key (file identifiers) each of them are thought to hold. Messages between Freenet and peer contain a 64-bit transaction ID which is randomly generated, a hops-to-live limit and a depth counter. Hops to live is set by the initiator of a message and decremented at each hop whereas the depth parameter is incremented at each hop. Thus Freenet has "smart" query system - the requests get routed to the correct peer by the incremental discovery.

Communication protocol

For initiating a transfer, a peer in the network identifies another by sending a Handshake Request message. If the peer it is trying to connect to

is active, it will respond with a Handshake Reply which would include the protocol version number that it is expecting. Handshakes are saved for a couple of hours so that further transactions between the same two peers, for this time period, can omit these set up steps. To obtain a file a Data Request message is sent which specifies the transaction ID and initial hops to

live limit. If the request is successful, the remote node will reply with a Data Found message containing the address of the node that supplied it and the length of the requested data. Then the data is sent in chunks with Data Chunk message.

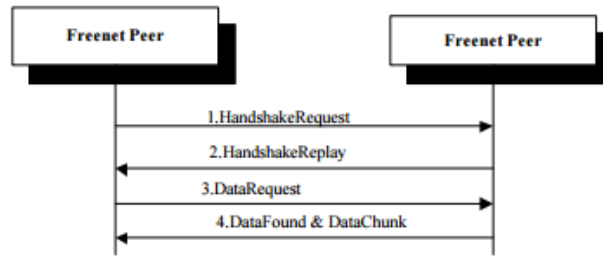


Figure 3: Logical flow of Freenet Messages [2]

3.2 GNUTELLA

The original Gnutella architecture is an example of an unstructured, purely decentralized system. A Gnutella network consists of a dynamically changing set of peers connected using TCP/IP and distinguished by its support for anonymity. Each peer acts as a client (an originator of queries), a server (a provider of file information), and as a router (a transmitter of queries and responses). Since it's completely decentralized, there is no coordination of activities in the network. It uses IP as its underlying network service. Each message has a unique identifier. Each host is equipped with a dynamic routing table of message identifiers. The response messages will contain the same ID as the original messages, the host will forward the message according to entry in the routing table, and similarly the unique identifier will be used to detect duplicate messages.

Communication protocol

Peers discover other peers on the system using 'Ping' (to discover hosts on the network) and 'Pong' (reply to a ping) messages. To query and get the file, a three step process is required. First step is to send a 'Query' message which specifies a set of files that the user is requesting. If any match is present, the second step is issued, a peer matching the query generates a 'QueryHit' message which shares information on acquiring the files. The third step is a establishing a direct connection to the peer using HTTP to retrieve the file (also called the Push step)

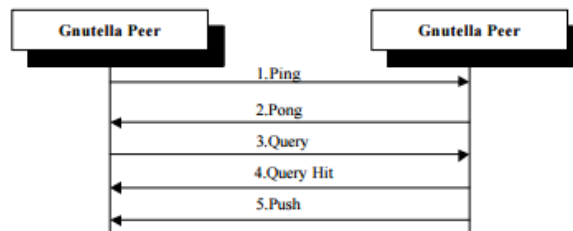


Figure 4: Logical flow of Messages in a Gnutella Network [2]

3.3 CHORD

A fundamental problem that confronts peer-to-peer applications is to efficiently locate the node that stores a particular data item. Chord is an example of a partially centralized structured network that addresses this problem. It provides support for

only one operation, mapping a key onto a node. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data item pair at the node to which the key maps [3]. Chord uses a variant of consistent hashing to assign keys to Chord node. Each Chord node needs routing information for only a few other nodes. (only $O(\log N)$ for an N -node system in the steady state), and resolves all lookups via $O(\log N)$ messages to other nodes. Chord maintains its routing information as nodes join and leave the system; with high probability each such event results in no more than $O(\log^2 N)$ messages[3]

Protocol [4]

As nodes enter the network, they are assigned unique IDs by hashing their IP address. Keys (file ids) are assigned to nodes as follows. Identifiers are ordered in an “identifier circle” modulo 2^m . Key k is assigned to the first node whose identifier is equal to or follows (the identifier of) k in the identifier space. This node is called the successor node of key. When a node n joins the network, certain keys previously assigned to n 's successor will become assigned to n . When node n leaves the network, all keys assigned to it will be reassigned to its successor. These are the only changes in key assignments that need to take place in order to maintain load balance. Queries for a given identifier are passed around the circle via these successor pointers until they first encounter a node that succeeds the identifier. This is the node the query maps to. In order to accelerate the process, Chord maintains additional routing information. This additional information is called a “finger table”, in which each entry i points to the successor of node $n+2^i$. In order for a node n to perform a lookup for key k , the finger table is consulted to identify the highest node n' whose id is between n and k . If such a node exists, the lookup is repeated starting from n' . Otherwise, the successor of n is returned. Using the finger table, lookups can be completed in time $\log(N)$. [4]

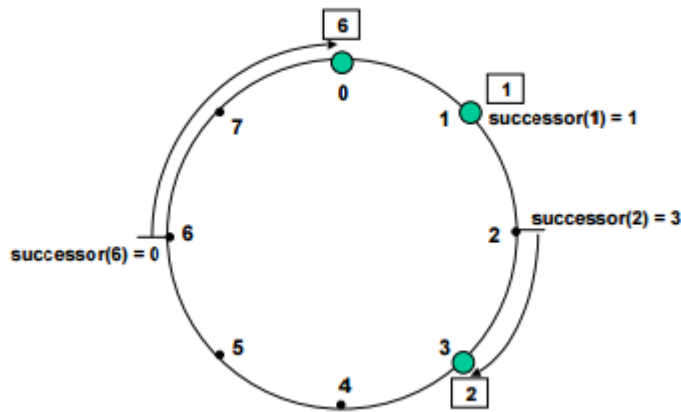


Figure 5: Chord node Identifier Circle[4]

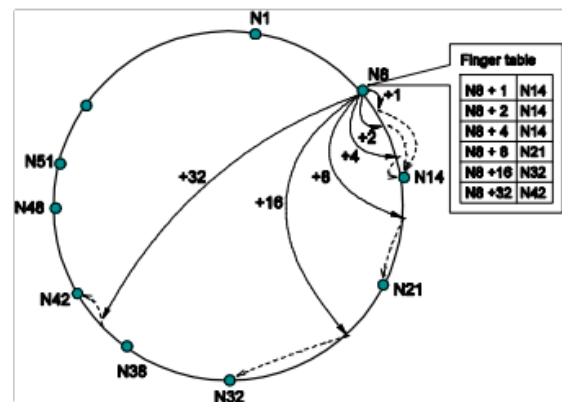


Figure 6: Chord Circle with Finger Table[4]

4 REFERENCES

1. Siu Man Lui and Sai Ho Kwok, 'Interoperability of Peer-To-Peer File Sharing Protocols' in ACM SIGecom Exchanges, Vol. 3, No. 3, August 2002
2. J. Lloret Mauri , G. Fuster , J. R. Diaz Santos and M. Esteve Domingo, 'Analysis and characterization of Peer-to-Peer Filesharing Networks' in <http://personales.upv.es/jlloret/pdf/icosmo2004.pdf>
3. Ion Stoica , Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan, 'Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications'
4. Stephanos Androutsellis-Theotokis, 'A Survey of Peer-to-Peer File Sharing Technologies' in Athens University of Economics and Business, Greece © Copyright 2002
5. http://www.webopedia.com/DidYouKnow/Internet/peer_to_peer.asp

Part II: Design of Peer To Peer File Sharing System

1 DESIGN METHODOLOGY

As per the problem statement given, a hybrid decentralized type of P2P Network with a central directory, and each peer having server and client functionality. In the following sections the protocols used to communication and design choices are documented. In this implementation, python programming language is being used.

1.1 CENTRAL DIRECTORY

- All communications in the Central Directory are handled using UDP Sockets.
- To include reliability to UDP, stop and wait protocol has been added. The peer sends fragmented packets to the directory. The directory in turn checks if the sequence number is as expected from the peer and sends an ACK as the next sequence number, so that the peer knows which indicates the next packet number the peer would have to send. If a duplicate packet is received it drops the packet and resends the acknowledgement
- Every time a request is received, the central directory processes it on a new thread with the threading functions `process_packet()`. This way it can receive multiple requests.
- A peer list is maintained and updated, in sync with a timer function.

1.1.1 Working and Functionality

- Request for file - A peer can request the central directory for a file and it will return a list of peers that own that particular file.
- Request to Exit – A peer can request to exit from the peer to peer network
- Update - A timer has been included in the central directory. An update message is received by the Central directory in sync with the timer, which updates the listing of the particular peer in the directory. This update message also acts as a ‘Keep Alive’ request in the approach followed, where if it doesn’t receive an update message, the peer is automatically removed from the network.
- Additionally the peer can add or delete sharing files, these changes are made to the update message

1.1.2 Message format

The requests from peer to central directory are appended with an integer which corresponds to the request type. In addition, messages from the peer have a sequence number field and a total number of packets field. This is included to implement the stop and wait protocol for reliability

REQUEST TYPE	FILENAME	SEQUENCE NUMBER	TOTAL NO. OF PACKETS
--------------	----------	-----------------	----------------------

Figure 7: Message format for Peer to Central Directory Communication

1.2 PEER

1.2.1 Working and Functionality

- The P2P Client sends a query to the central directory to see all the directory listing of a particular file. A function `send_udp_data()` fragments the message and sends it over a udp socket to the central directory and waits for an acknowledgement from the Directory. If it doesn't receive an ACK in a given time interval, it times out and retransmits the packet.
- The timeout interval is calculated based on the equations,
$$EstimateRTT = (1-\alpha)EstimateRTT + \alpha SampleRTT$$
$$DevRTT = (1 - \beta)DevRTT + \beta |SampleRTT - EstimateRTT|$$
The timeout is handed in a timer function
- Once the query for the file is sent and the CD returns list of the peers to connect to, it inputs a peer, and the transfer process is started using a HTTP socket.
- The request to the transient P2P server is sent in http format
- The transient P2P server responds with a http response – 200 OK to start file download
- The http server is multi-threaded to handle multiple requests

1.2.2 Message format

The 'GET' string is appended to the requested inputted by the user to initiate an HTTP get request. The request line is,



Figure 8: Message format for P2P Client to P2P Server Communication

Opening an http socket takes care of the rest of the field of the HTTP request – like version number, header lines etc.

2 PSEUDO CODE

2.1 CENTRAL DIRECTORY

Open UDP Socket

Start the timer

Wait for Client Connections

If received data from client

 send the buffer to the socket

 If the old buffer received is not equal to the new buffer received

 Find the num_packets

 Find the seq_packets

 If num_packets is equal to number of seq_packets

 Start the thread : ProcessPackets

Thread : ProcessPackets

Split the data obtained

If Request for UPDATE/CONNECT

```

        Add a new peer to the peer list
    else if Request for FILE LOOK UP
        If requested peer is present:
            send the peers information
        else
            send the error data
    else if Request for DELETION
        Remove the peer from the list

```

2.2 PEER

```

Start HTTP Server over TCP Socket
Inform the Central Directory that the Peer is Up and Running
Start Timer for Keep Alive/UPDATE
If Request for File
    Send a UDP_Request to server
    Get the file from the server as HTTP-RESPONSE

else if Request to share/add file
    Add file to the sharing list

else if Delete sharing of file
    If filename present in the list
        remove the filename
    else
        return

else if Request for list of files
    return the list of files in the sharing list

else if Request for exit of peer
    close the udp connection with the server
    shutdown the http connection
    exit

else if Request peer entries in the central directory
    return list of peers

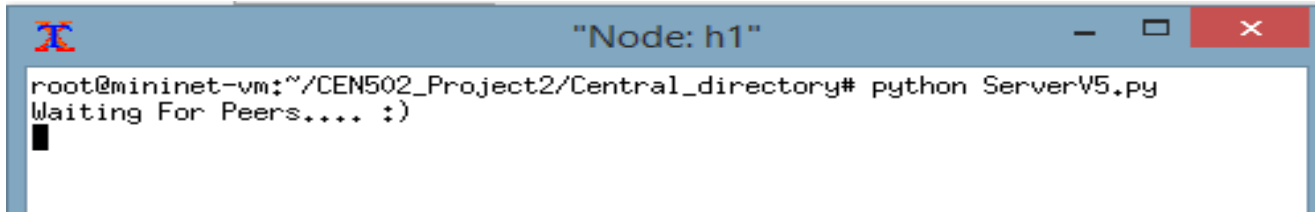
else
    return

```

3 TEST AND SAMPLE OUTPUT

1. Successful start of Central Directory and Peers

Starting central directory



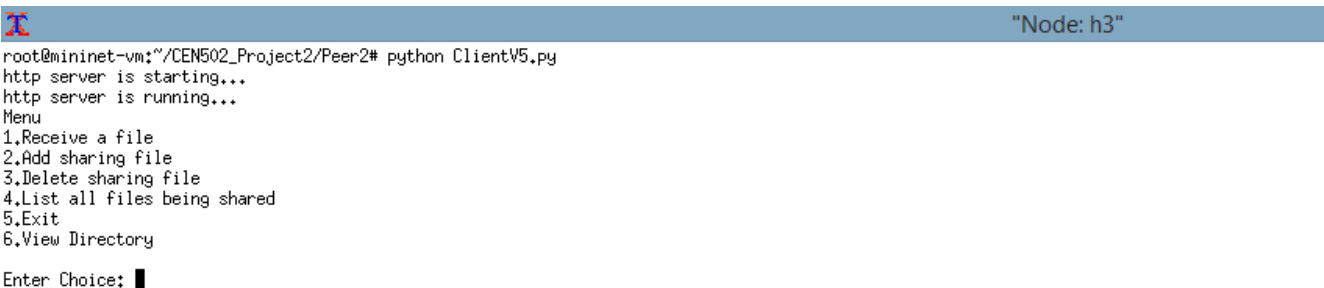
```
root@mininet-vm:~/CEN502_Project2/Central_directory# python ServerV5.py
Waiting For Peers.... :)
█
```

Starting Peer 1



```
root@mininet-vm:~/CEN502_Project2/Peer1# python ClientV5.py
http server is starting...
http server is running...
Menu
1.Receive a file
2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory
Enter Choice: █
```

Starting Peer 2



```
root@mininet-vm:~/CEN502_Project2/Peer2# python ClientV5.py
http server is starting...
http server is running...
Menu
1.Receive a file
2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory
Enter Choice: █
```

Starting Peer 3



```
root@mininet-vm:~/CEN502_Project2/Peer3# python ClientV5.py
http server is starting...
http server is running...
Menu
1.Receive a file
2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory
Enter Choice: █
```

TEST 1: Adding of a file to Directory

UPDATE - Sharing/Adding File new.txt [H3] to the Directory

```
root@mininet-vm:~/CEN502_Project2/Peer2# python ClientV5.py
http server is starting...
http server is running...
Menu
1.Receive a file
2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory

Enter Choice: 2
Enter the file you want to share:new.txt
Menu
1.Receive a file
2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory

Enter Choice: 4
Getting the list of files...
1 10.0.0.3 80 new.txt
"
```

View Directory Contents, Directory UPDATE SUCCESS!!

```
root@mininet-vm:~/CEN502_Project2/Peer1# python ClientV5.py
http server is starting...
http server is running...
Menu
1.Receive a file
2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory

Enter Choice: 6
[[[], ['10.0.0.4', '80', 1446419619,305229], ['10.0.0.2', '80', 1446419620,260388], ['10.0.0.3', '80', 'new.txt', 1446419623,079928]]
('10.0.0.1', 7777)
Menu
"
```

Wireshark Trace - Peer 1, Peer2, Peer3 interacting with Central Directory over UDP

Wireshark 1.10.6 (v1.10.6 from master-1.10)

Filter: udp

No.	Time	Source	Destination	Protocol	Length	Info
332	6.580757000	10.0.0.1	10.0.0.2	UDP	62	Source port: cbt Destination port: ddi-udp-1
537	9.292658000	10.0.0.3	10.0.0.1	UDP	70	Source port: krb524 Destination port: cbt
538	9.293331000	10.0.0.1	10.0.0.3	UDP	70	Source port: cbt Destination port: krb524
670	12.487872000	10.0.0.4	10.0.0.1	UDP	62	Source port: ddi-udp-1 Destination port: cbt
671	12.488470000	10.0.0.1	10.0.0.4	UDP	62	Source port: cbt Destination port: ddi-udp-1
689	13.606324000	10.0.0.2	10.0.0.1	UDP	62	Source port: ddi-udp-1 Destination port: cbt
690	13.613081000	10.0.0.1	10.0.0.2	UDP	62	Source port: cbt Destination port: ddi-udp-1
709	16.308799000	10.0.0.3	10.0.0.1	UDP	70	Source port: krb524 Destination port: cbt
710	16.309437000	10.0.0.1	10.0.0.3	UDP	70	Source port: cbt Destination port: krb524
834	19.502112000	10.0.0.4	10.0.0.1	UDP	62	Source port: ddi-udp-1 Destination port: cbt
835	19.503046000	10.0.0.1	10.0.0.4	UDP	62	Source port: cbt Destination port: ddi-udp-1
916	20.630876000	10.0.0.2	10.0.0.1	UDP	62	Source port: ddi-udp-1 Destination port: cbt
917	20.631321000	10.0.0.1	10.0.0.2	UDP	62	Source port: cbt Destination port: ddi-udp-1

Frame 689: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0

Linux cooked capture

Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 10.0.0.1 (10.0.0.1)

User Datagram Protocol, Src Port: ddi-udp-1 (8888), Dst Port: cbt (7777)

Data (18 bytes)

```
0000 00 00 00 01 00 06 1e d0 fa c6 42 76 01 00 08 00 .....Bv....
0010 45 00 00 2e 0d 31 40 00 40 11 19 8c 0a 00 00 02 E....1@. @....
0020 0a 00 00 01 22 b8 1e 61 00 1a 2d 01 31 20 31 30 .....a...1 10
0030 2e 30 2e 30 2e 32 20 38 30 20 20 31 20 31 20 31 0.0.2 8 0 1 1
```

TEST 2: Delete a file entry from the Central Directory

Request for Deletion of file recv.txt [h2] from Central Directory

```
"Node: h2"

Enter Choice: 6
[[[], ['10.0.0.4', '80', 1446420657,719606], ['10.0.0.2', '80', 'new.txt', 'recv.txt', 'file1.txt', 1446420659,088473], ['10.0.0.3', '80', 'f1.txt', 1446420660,358935]]
('10.0.0.1', 7777)
Menu
1.Receive a file
2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory

Enter Choice: 3
Enter the file you want to Delete:recv.txt
DELETE COMPLETE!!!
Menu
1.Receive a file
2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory

Enter Choice: █
```

Check for update by viewing the entries in the Central Directory [h3]

```
"Node: h3"

Enter Choice: 6
[[[], ['10.0.0.4', '80', 1446420720,85509], ['1', '10.0.0.2', '80', 'new.txt', 'file1.txt', 1446420722,258604], ['10.0.0.3', '80', 'f1.txt', 1446420723,649439]]
('10.0.0.1', 7777)
Menu
1.Receive a file
2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory
```

Check for update by viewing the entries in the Central Directory [h2]

```
"Node: h2"

2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory

Enter Choice: 3
Enter the file you want to Delete:recv.txt
DELETE COMPLETE!!!
Menu
1.Receive a file
2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory

Enter Choice: 6
[[[], ['10.0.0.4', '80', 1446420896,264278], ['1', '10.0.0.2', '80', 'new.txt', 'file1.txt', 1446420897,753142], ['10.0.0.3', '80', 'f1.txt', 1446420899,461182]]
('10.0.0.1', 7777)
Menu
```

TEST 3: Request for file

Host 4 requests for file download from host2 – 200 OK SUCCESS!!

```
root@mininet-vm:~/CEN502_Project2/Peer3# python ClientV5.py
http server is starting...
http server is running...
Menu
1.Receive a file
2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory

Enter Choice: 6
[[[]], ['1', '10.0.0.2', '80', 'new.txt', 'file1.txt', 'recv.txt', 1446421931,274489], ['10.0.0.3', '80', 'new.txt', 1446421932,645776], ['10.0.0.4', '80', 1446421933,628407]]
('10.0.0.1', 7777)
Menu
1.Receive a file
2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory

Enter Choice: 1
Enter the file you require:new.txt
Processing the Request
2 new.txt
Retrieved sources of files
Choose a peer from the list to download new.txt
Choice 0
10.0.0.2 80
Choice 1
10.0.0.3 80
Enter Choice: 10.0.0.2 80
Request File from peer: ['10.0.0.2', '80'] ('10.0.0.1', 7777)
MESSAGE 2 new.txt
GET
new.txt
(200, 'OK')
FILE DOWNLOAD COMPLETE!!!
Menu...
```

Wireshark - HTTP Requests and Response between H2 and H4 over TCP

The image shows a Wireshark packet capture of an HTTP transaction between host 2 (10.0.0.2) and host 4 (10.0.0.4). The filter is set to 'ip.addr == 10.0.0.2'. The packet list shows several TCP and HTTP packets. The selected packet is a GET request for 'new.txt' from host 2 to host 4, which is responded to by a 200 OK message from host 4.

No.	Time	Source	Destination	Protocol	Length	Info
1153	39.872855000	10.0.0.4	10.0.0.2	TCP	68	46173 > http [ACK] Seq=68 Ack=7354 Win=44032 Len=0 TSval=26614409 TSecr=26614408
1154	39.872880000	10.0.0.4	10.0.0.2	TCP	68	46173 > http [ACK] Seq=68 Ack=13146 Win=55296 Len=0 TSval=26614409 TSecr=26614408
1155	39.873108000	10.0.0.2	10.0.0.4	HTTP	14548	Continuation or non-HTTP traffic
1156	39.873117000	10.0.0.2	10.0.0.4	HTTP	14548	Continuation or non-HTTP traffic
1157	39.873141000	10.0.0.4	10.0.0.2	TCP	68	46173 > http [ACK] Seq=68 Ack=27626 Win=84480 Len=0 TSval=26614409 TSecr=26614409
1158	39.873146000	10.0.0.4	10.0.0.2	TCP	68	46173 > http [ACK] Seq=68 Ack=42106 Win=113664 Len=0 TSval=26614409 TSecr=26614409
1159	39.873194000	10.0.0.2	10.0.0.4	HTTP	14548	Continuation or non-HTTP traffic
1160	39.873220000	10.0.0.4	10.0.0.2	TCP	68	46173 > http [ACK] Seq=68 Ack=56586 Win=140288 Len=0 TSval=26614409 TSecr=26614409
1161	39.873201000	10.0.0.2	10.0.0.4	HTTP	1516	Continuation or non-HTTP traffic
1162	39.873226000	10.0.0.4	10.0.0.2	TCP	68	46173 > http [ACK] Seq=68 Ack=58034 Win=139264 Len=0 TSval=26614409 TSecr=26614409
1163	39.873210000	10.0.0.2	10.0.0.4	HTTP	860	Continuation or non-HTTP traffic
1168	39.876687000	10.0.0.4	10.0.0.2	TCP	68	46173 > http [FIN, ACK] Seq=68 Ack=58827 Win=142336 Len=0 TSval=26614410 TSecr=26614409
1169	39.876797000	10.0.0.2	10.0.0.4	TCP	68	http > 46173 [ACK] Seq=58827 Ack=69 Win=29184 Len=0 TSval=26614410 TSecr=26614410

Frame 1155: 14548 bytes on wire (116384 bits), 14548 bytes captured (116384 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 10.0.0.4 (10.0.0.4)
Transmission Control Protocol, Src Port: http (80), Dst Port: 46173 (46173), Seq: 13146, Ack: 68, Len: 14480
Hypertext Transfer Protocol

0040 01 96 1a 09 76 2c 20 6d 6e 6a 66 6b 65 77 6c 8b ...v, m rjfkewl;
0050 64 2e 66 76 20 6d 66 64 6b 77 3b 6c 2e 66 2c 76 d.fv mfd kw,l f,v
0060 66 6b 64 2e 66 2c 67 76 6b 65 64 2e 66 76 20 6d fkd,f,gv ked,fv m
0070 66 6b 65 64 2e 0d 0a 68 75 66 65 6a 64 63 6e 20 fked, h ufejden
0080 66 76 62 67 68 66 72 65 6a 69 77 64 70 6c 3b 2c fvbghfrc jwdpl;
0090 63 20 76 6d 6a 6a 66 6b 65 6f 64 3b 2e 76 2c 20 c vnmjfk eed,v,
00a0 6d 6e 66 6a 65 6b 6f 64 3b 2e 66 76 2c 20 6d 6a mnfjekod ;fv, mn
00b0 6a 66 6b 65 77 6c 3b 64 2e 66 76 20 6d 66 64 6b jfkewl;d .fv mfdk
00c0 77 3b 6c 2e 66 2c 76 66 6b 64 2e 66 2c 67 76 6b w,l f, of kd,f,gvh
00d0 65 64 2e 66 76 20 6d 66 6b 65 64 2e 0d 0a 68 75 ed,fv mf ked, h
00e0 66 65 6a 64 63 6e 20 66 76 62 67 68 66 72 65 6a fejden f vbghfrc
00f0 69 77 64 70 6c 3b 2c 63 20 76 6d 6a 66 6b 65 iwdpl;c vnmjfk
0100 6f 64 3b 2e 76 2c 20 6d 6a 66 6a 65 6b 6f 6d 8b od,v, m rjfkewl;

File Download SUCCESS!!

```

"Node: h4"
root@mininet-vm:~/CEN502_Project2/Peer3# ls
ClientV5.py  new.txt
root@mininet-vm:~/CEN502_Project2/Peer3# ls -lrt
total 68
-rw-r--r-- 1 root root 7189 Nov  1 15:12 ClientV5.py
-rw-r--r-- 1 root root 58710 Nov  1 15:52 new.txt
root@mininet-vm:~/CEN502_Project2/Peer3# █
```

TEST4: ERROR CODES

Request for a file from a Peer which does not have the file – 408 file not found

```

"Node: h4"
Enter Choice: 6
[[[], ['10.0.0.4', '80', 1446422879.862928], ['10.0.0.3', '80', 'f1.txt', 'f3.txt', 1446422880.12232], ['10.0.0.2', '80', 'new.txt', 'file1.txt', 1446422880.944059]]
('10.0.0.1', 7777)
Menu
1.Receive a file
2.Add sharing file
3.Delete sharing file
4.List all files being shared
5.Exit
6.View Directory
Enter Choice: 1
Enter the file you require:new.txt
Processing the Request
2 new.txt
Retrieved sources of files
Choose a peer from the list to download new.txt
Choice 0
10.0.0.2 80
Enter Choice: 10.0.0.3 80
Request File from peer ['10.0.0.3', '80'] ('10.0.0.1', 7777)
MESSAGE 2 new.txt
GET
new.txt
(404, 'file not found')
```

4 CONCLUSION

The P2P file sharing application was implemented using a central directory and as per the requirements in the problem statement and was tested on a three peer system. The results were as expected and shown in Section 3.

5 REFERENCES

2. <https://docs.python.org/2/tutorial/classes.html#a-first-look-at-classes>
3. <https://docs.python.org/2/howto/sockets.html>
4. Beej's Guide to Network Programming - Using Internet Sockets
5. James F. Kurose and Keith W. Ross, 'Computer Networking A Top-Down Approach ' Sixth Edition
6. stackoverflow.com