Advanced Operating Systems
CS-550

Programming Assignment 2

A Simple Gnutella-style P2P File Sharing System

Group Members:

Kunal Dhande
A20369862

kdhande@hawk.iit.edu

Mohit Kulpe
A20372614

mkulpe@hawk.iit.edu

Content Table

1. Introduction

1.1. Problem Statement

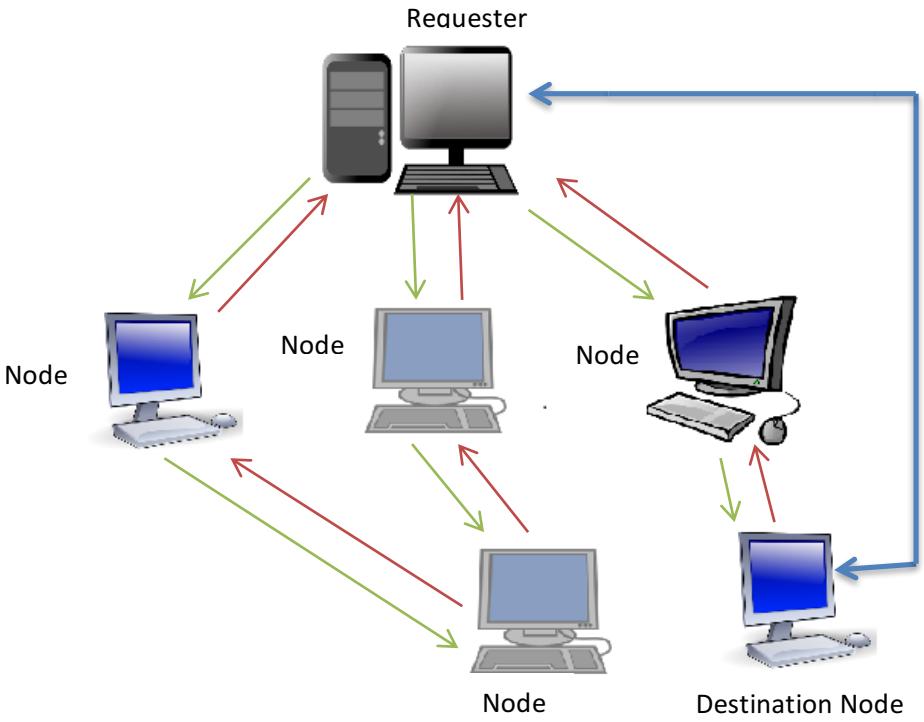
A simple Gnutella-style peer-to-peer (P2P) system to be designed. Each peer should be both a server and a client. As a client, it provides interfaces through which users can issue queries and view search results. As a server, it accepts queries from other peers, checks for matches against its local data set and responds with corresponding results. search is done in a distributed manner. Each peer maintains a list of peers as its neighbor. Whenever a query request comes in, the peer will broadcast the query to all its neighbors in addition to searching its local storage.

1.2. System Introduction

This document provides a detailed design of the Distributed File Sharing System. It provides an overview of how different components in the system interact with each other. In this Peer to Peer File Sharing system searching is done in a distributed manner. As a Client user passes a query on other end as a Server it accepts query and searches in local database responds to that respected query.

2. Architecture

A decentralized peer-to-peer system, consisting of hosts connected to one another and running software that implements the Gnutella protocol. The connection of individual nodes forms a network of computers exchanging request & response in form of queries & response to queries.



Note:

: Query

: Query Response

: Download

Figure 1

2.1 Working of system

1) A node that wishes to participate in the network must join the Gnutella network by Connecting to any existing node.

2) Requester passes request of required file in form of Query to neighboring nodes as shown in figure.

3) Then the intermediate nodes in network further searches for data in local storage and broadcast that query to multiple other nodes in network as shown in figure.

4) After finding destination node where the requested data is present, Requester gets positive reply in form of Query response else Requester gets negative reply in form of Query response as shown in figure.

5) Then that destination node acts as server and requester acts as client, further requester downloads required file from that destination node as shown in figure.

3. Implementation

3.1 Requester

initialized the P2P network, a requester peer searches for files by issuing a query. The query is sent to all neighbors. Each neighbor looks up the specified file using a local index and responds with a *queryhit* message in the event of a hit.

```
    public ArrayList<PeerDetails> search(String searchFileName) throws InterruptedException {
        List<Thread> threads = new ArrayList<Thread>();
        ArrayList<PeerDetails> resultsFromPeer = new ArrayList<PeerDetails>();
        ArrayList<PeerDetails> neighboringPeers = new ArrayList<PeerDetails>();
        ArrayList<NeighborHandler> neighboringThreads = new ArrayList<NeighborHandler>();

        // Get Neighboring peers
        getNeighboringPeers(neighboringPeers, peerID);

        // Generate unique message id
        ++msgIDConuter;
        String msgId = System.currentTimeMillis()+ "-" + peerID + "-"+ msgIDConuter;

        // Loop through all the neighbor peers
        for (PeerDetails neighboringPeer: neighboringPeers) {

            NeighborHandler connection = new NeighborHandler(searchFileName,
                    neighboringPeer.hostIP, neighboringPeer.portNo, peerID,
                    neighboringPeer.peerID, msgId,TTL);
            Thread threadInstance = new Thread(connection);
            threadInstance.start();

            // Add connection thread instances
            threads.add(threadInstance);
            neighboringThreads.add(connection);
        }

        // Wait until all child threads finish execution
        for (Thread thread: threads)
            thread.join();

        // Get hit query result from all the neighbor peers
        System.out.println("\n");
        for (NeighborHandler neighboringThread: neighboringThreads) {
            HitQueryResult hitQueryResult = (HitQueryResult) neighboringThread.getValue();
            if (hitQueryResult.searchResults.size() > 0)
                resultsFromPeer.addAll(hitQueryResult.searchResults);
        }

        return resultsFromPeer;
    }
```

3.2 Destination Node

A *queryhit* message is sent back to the original sender following the reverse path. The message ID is used for this purposes. *Obtain* is achieved by sending a direct download request to a peer that sends a *queryhit* message.

```
public synchronized byte[] obtain(String filename) throws RemoteException {
    // Remote method for downloading the file
    byte[] fileBytes = null;
    String fullFileName = sharedDirectory + "/" + filename;
    try {
        fileBytes = new byte[(int) (new File(fullFileName)).length()];
        BufferedInputStream input = new BufferedInputStream(new FileInputStream(fullFileName));
        // get the data in bytes
        input.read(fileBytes, 0, fileBytes.length);
        input.close();
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
    return fileBytes;

}
```

4.Future Enhancements

1. We should pass full name of file which should be searched/download, hence our future enhancement will be to apply search algorithms to find the partial matches.

2. Network efficiency and scalability can be improved by adding some nodes in network with high bandwidth.

5. Assumptions and Constraints

1. If the node which has file is closed, then the other peer which wants to download that file will not be able to connect and download the file.

2. Requester should type exact file name that should be searched else the query response.

3. A node in a network with low bandwidth will not perform efficiently as compare to other nodes with high bandwidth, hence will degrade overall performance.