P2P Network File Transfer

Project 1

Readme and Design Document

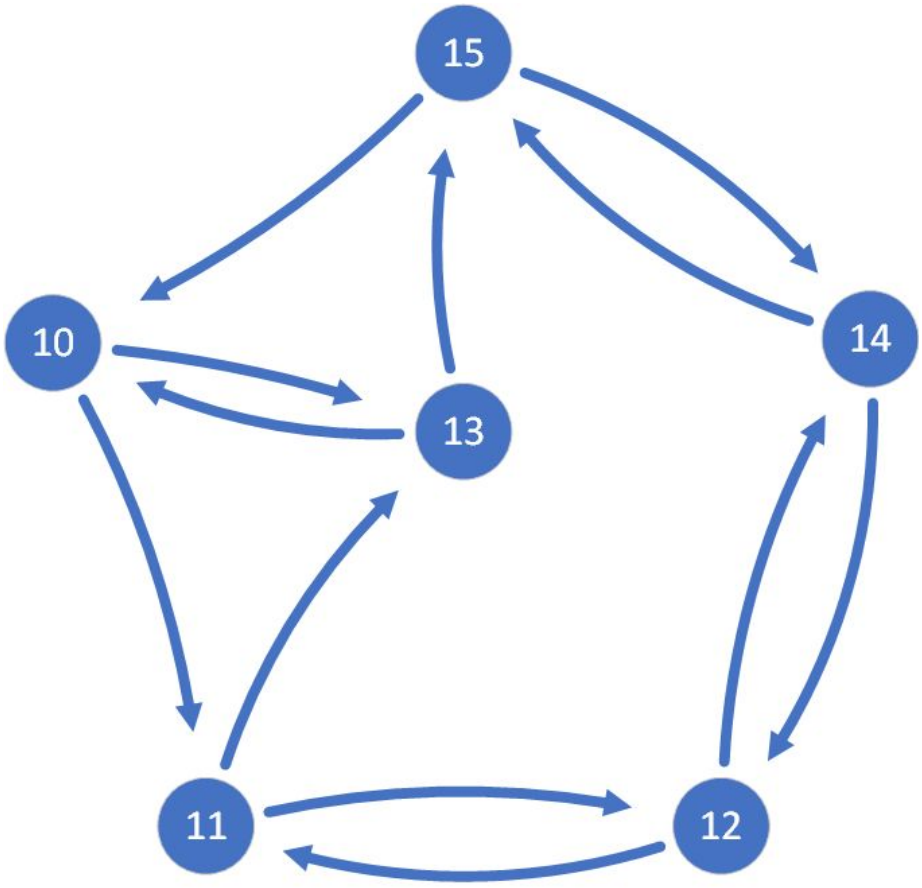
EECS 325

By Kris Zhao

1. Network Hierarchy

1.1 Peer connections:

To test different loops, and necessity of peer forwarding, I set up the five hosts like so:



Each host listens on port 51,045 for request connections and on port 51,050 for file request connections.

Each host will also have the following files for requesting*:

Peer name:	Files:
eecslab-10	11-0.txt 1184-0.txt 1342-0.txt 135-0.txt

	1400-0.txt
eecslab-11	158-0.txt 16-0.txt 1952-0.txt 219-0.txt 244-0.txt
eecslab-12	2554-0.txt 2591-0.txt 2600-0.txt 2701-0.txt 28054-0.txt
eecslab-13	2814-0.txt 3090-0.txt 35-0.txt 4300-0.txt 43-0.txt
eecslab-14	56571-0.txt 56572-0.txt 56576-0.txt 56579-0.txt 56581-0.txt
eecslab-15	56582-0.txt 56584-0.txt 6130-0.txt 74-0.txt 76-0.txt 829-0.txt 84-0.txt 98-0.txt

*I believe that the hosts should have these files. It is possible, even though I double checked, that a slight typo was made

2. Assumptions

2.1 User behavior

For the purposes of simplicity, I assumed that the user would act in accordance to the program. This included reasonable inputs as well as exiting. There is a little amount of user input error handling

included. In terms of exiting, I assume the user will use the leave / exit commands to exit the program and not forcefully disconnect.

3. Design

3.1 Possible Issues and solutions

3.1.1 Problem 1: Unique IDs

It was necessary to have a unique ID for queries across all of the peers. Because of this, I wanted to utilize an ID containing the peer hostname / IP added onto the file the host was looking for. This would uniquely identify the queries and help trace back the location for responses. While this prevents the user from requesting the same file twice, I believe this can prevent needless loading of the P2P network.

3.1.2 Problem 2: Query Loops / Broadcast Storms

The unique IDs allowed for the prevention of query loops in both the requests and responses. Given an interconnected network, it's very easy for a peer to see the same request twice. As a result, each peer will remember the requests it sent out, the requests it has forwarded, and the responses to each request. Then, whenever it receives any kind of query, it will check to make sure it hasn't seen / forwarded it before. This will prevent endless propagations of queries.

3.2 General Structure and Components:

3.2.1 Welcomers:

For the P2P network, we required two welcome handlers. One for querying and another for the file transfer. As such, I created two welcomers which would handle each of these connections. Utilizing threads, this would allow the P2P network to listen in the background without blocking the main program.

3.2.2 Handlers:

Handlers have a similar task to Welcomers as they will also handle a single socket. For simplicity, I separate the handlers into a receiver and a sender. The receiver would be created by the welcome socket and will receive queries and send responses. The sender on the other hand, is created on command **connect**. They are responsible for sending out the requests and then receiving the responses.

3.2.3 Data-structures:

The design of my program hinged on a couple specific data structures. One handled remembering the queries and another to store them.

3.2.3.1 Query

The Query datatype was created with the primary function of passing query data back and forth. As such, it would have all the information for the IDs, previous IPs, and filenames... etc. This also aids with response forwarding so that, after a response is received, it can be matched to whichever peer sent the request through ID.

3.2.3.2 Maps

The maps were used to help with storing the actual queries in the peer. This would utilize a string as the key and that actual Query as the object. This allowed for simple checks to determine whether a query has been received before, as well as getting the request query and it's source.

3.2.4 Heartbeats:

For the heartbeat, I read the specifications to mean that the peer should send heartbeats to any other peer it has connected to. In other words, the peers specified in the neighbors file. They are not responsible for sending heartbeats to incoming connections. Here, the heartbeat will be sent into the TCP connection as a string by utilizing a timer at an interval of once every 30 seconds. The receiving end has a timer counting down from 60 seconds to close the connection unless it receives a heartbeat.

3.2.5 Printing

As output and information for the user, there will be statuses printed as mentioned below:

- Whenever the peer program starts up (and before any commands are given).
- When the peer initiates connections to any other host and whether the connection failed or succeeded.
- Whenever a peer receives a connection from another peer.
- When the peer sends out a request query to the peers it's connected to (Query flooding).
- When the peer receives a query, as well as the result (Having the file, dropping request, or forwarding query?)
- Whenever you send a heartbeat message
- Whenever you receive a heartbeat from a neighbor
- When a incoming connection times out due to a missed heartbeat
- When the peer sends out a request for a file from another peer.
- When the peer has a file requested from it (and when it was completed).