Table of Contents

1. Introduction

Peer to Peer Content System

A common content delivery system on the internet uses a client-server architecture where a client seeks to download a file contained on the server. This architecture can provide more granular security and access controls but can also be a bottleneck for when many clients are attempting to retrieve the same content. A peer to peer network allows for clients to host the content available to them and download content from other clients. This reduces the client to server bandwidth since clients connect to each other to download and only use the server as a way to list and discover where to download content.

Socket Programming

Socket programming could be used to build this system through the UDP and TCP protocols. A TCP protocol involves a three-way handshake before transferring data. TCP is a connection-oriented protocol, whereas UDP is a connectionless protocol. The speed for TCP is slower while the speed of UDP is faster. These properties make TCP a robust and deterministic protocol for file transfer, making it a great choice for the client to client connection. Due to its speed, the UDP protocol can be used by the client to update the index server as soon as file contents in each client have changed.

2. Technical Description

Client

The client handles direct communication with the end user, who decides on the content to register and download. The client also acts as a content server — hosting a piece of content for download — or a content client, who requests to download a specific piece of content. At each runtime, the user provides a peer name to the client that acts as the unique identifier per running client. The client allows the user to perform the following functionalities:

Content Registration (R)

The user can register any file stored locally on the host machine to the index server as available for download. To do this, the user supplies the name of the file, and the client sends an R packet to the server to request for registration. If the server response packet is an A type, the content has been acknowledged, and the content is added to a local list of registered content for use in the (L) functionality. The client also opens a TCP connection, with a unique port assigned to the content, that listens for incoming file download requests for said content. The port listening and file transfer is non-blocking since it is handled by a child process.

Content Deregistration (T)

The user can deregister any piece of registered content. The client will send a T type packet to the server with the peer name and content name to request a deregistration. The content is also deleted from the list of local content.

List Local Content (L)

The user can request to view all the local content that has been registered to the index server and is actively ready for download. This lists the content name and the associated port.

List Index Server Content (O)

The user can request to view all content registered on the index server. The client will send an O type packet to the server as a request, and will receive and display the incoming list to the user. The list includes all uniquely registered content across all clients connected to the server.

Download Content (D)

The user can request to download content from a registered content server by supplying the name of the content. The client will send an S type packet to the server with the peer name and the content requested. If verified, the server will supply an S type packet with the address of the content server. The client then establishes a TCP connection with the content server, and sends a D type packet with the peer name and requested content name.

The content server accepts the incoming request then looks for the requested content locally. When found, the content server sends a series of C type packets that transmit the contents of the requested file over the established TCP connection. Once complete, the content server closes the TCP connection.

The content client, once the file has been downloaded locally, automatically registers the content to the index server as ready for download, and follows the same steps as outlined in the (R) section.

Quit (Q)

The user uses the (Q) command to deregister all locally registered content from the index server, and exit the application. The content deregistration follows the same steps as outlined in the (T) section.

Server

The server's main purpose is to keep an updated index of content corresponding to where other clients may be able to find it. At each runtime, the server initializes and waits for clients to send information to populate it. The server has two main data structures: the queue and the set. The queue is an atomic list of contents mapped to its respective providers where new items are added in the back of the list and searching begins at the front. The *queue* is implemented using an array of custom content structs holding the content's name as well as the provider peer's host, port, and peername. The *set* is responsible for holding the name of all distinct content and is implemented using an array. The server can handle a number of client requests:

Register Content

When the R type packet is sent from the client (the peer), indicating a request to register content. The server will validate that the content not is already provided by the requesting peer and if the peer's selected name has not already been registered by another peer (with another host address). If the packet passes validation, the server will push the content into the end of the *queue*. If the content is unique and not yet listed on the server, the server will push the name of the content into the *set*. After these operations are complete, the server sends an A type packet to the peer to acknowledge the registration. If either of the validations fail, the server does not make any *queue* or *set* operations and returns an E type packet with an error message.

List Server Content

The user can request to view all content registered on the index server. The client will send an O type packet to the server as a request, signaling a request for server content, the server will send back an O type packet with a list of everything in the *set*. Recall that the *set* only includes distinct content and duplicates of content provided by multiple servers will only be counted once.

Content Download Address Request

A client can request a download address to a piece of content they wish to download. The client sends an S type packet along with the content requested. When this S type packet is received, the server looks through the queue to find the selected item. If an item is found, the address (host and port) are included in an S type packet back to the peer. If it is not found, the server returns an E type packet with an error message.

Implementation Detail: Load Balancing

In order to balance the request and download load between peers who have provided frequently downloaded and identical content, the index system implements a simple round robin algorithm within the queue. When a peer requests content and the server finds it in the queue, the item is copied to send back to the peer but is taken out and placed at the end of the queue. In C, this is completed by keeping a pointer to the end of the array, shifting each element (n) of the array to the (n-1)th spot after the element of interest is found, and finally appending the element of interest to the back of the queue. Because of the first found heuristic, if multiple copies of the content are provided by different peers, the search will always return the least downloaded peer.

Deregister Content

Similarly to content registration, peers can request to have their content de-registered from the server to stop other peers from being able to download from them. This is done both during the explicit de-register instruction or if the peer loses connection to the index server (or quits). The client sends a T type packet specifying the content to be deregistered. After the server receives this packet, the queue is searched for the content. If the content is found, the content is deleted from the queue and all the elements are shifted up. The queue is searched again for the same content name to check if any other provider peers offer the identical content. If not, the content is taken out of the set. After this is done, the server is sent an A type packet to acknowledge the operation. If the file was not found in the queue or there was an error in processing, the client is sent back an E type packet with a relevant error message.

3. Observations and Analysis

The following section documents a typical use case for the ecosystem from a user standpoint. All technical details have already been explained in section 2.

Initially, the server is started up and waits for incoming packets. User 1 starts an instance of the client service and provides the address to the index server, as well as a unique name, Mai. The client then provides a list of functions that the user can do, and prompts for a response, see Figure 3.1.

Figure 3.1: Client startup

```
============================
Welcome to the P2P Network!
============================
Please enter a username:
Mai
Welcome Mai! Choose a task:
    R: Register content to the index server
    T: De-register content
    D: Download content
    O: List all the content available on the index server
    L: List all local content registered
    Q: To de-register all local content from the server and quit
```

Mai has a piece of content, *lorem*, she wants to register, and thus runs (R) and supplies the content name. The client runs a validation sequence to ensure that the requested content is available locally, see Figures 3.2 and 3.3.

Figure 3.2: (R) functionality with invalid content

```
R
Enter a valid content name, 9 characters or less:
shrek
Error: File shrek does not exist locally
```

Figure 3.3: (R) functionality with valid content

```
R
Enter a valid content name, 9 characters or less:
lorem
The following content has been successfully registered:
    Peer Name: Mai
    Content Name: lorem
    Host: 0
    Port: 49279
```

If valid, the client sends the data to the server for registration. Mai can then run (O) to view that her content has been successfully registered, as well as run L to view locally registered content, see Figure 3.4 and 3.5.

Figure 3.4: (O) functionality

```
O
The Index Server contains the following:
    0: lorem
```

Figure 3.5: (L) functionality

```
L
List of names of the locally registered content:
Number          Name            Port            Socket
0               lorem           49279           4
```

Meanwhile, User 2, Jeff, and User 3, Hamzah, each start up their own clients. Jeff wants to download *lorem*, and does so using the (D) functionality, see Figure 3.6. Notice that the address of the content server that hosted *lorem* is the same as the one generated when Mai registered *lorem* as downloadable content. After a successful download, the client automatically registers the downloaded content to the index server, making Jeff another peer that a client can connect to to download *lorem.*

Figure 3.6: (D) functionality

```
D
Enter the name of the content you would like to download:
lorem
Trying to download content from the following address:
   Host: 0
   Port: 49279
Successfully downloaded content:
   Content Name: lorem
The following content has been successfully registered:
    Peer Name: Jeff
    Content Name: lorem
    Host: 0
    Port: 42741
```

Hamzah has a piece of local content, *shrek*, that he wants to register. He follows the same steps as Mai to register his content, and once registered, Jeff runs (O) to view that the content is available for download, Figure 3.7. Hamzah wants to close down his client and disconnect from the ecosystem so he runs (Q). (Q) deregisters all local content and quits the client application, and when any other Users check the index server, they will notice that *shrek* is now unavailable for download.

Jeff also quits the application and has his content, *lorem,* automatically deregistered. Since Mai is still hosting *lorem* however, it is still present when a User runs the (O) command.

Figure 3.7: Viewing content that is not hosted locally

```
O
The Index Server contains the following:
    0: lorem
    1: shrek
```

Overall the application works as intended with various validation sequences and limitations implemented to ensure that edge cases have been considered and accounted for to prevent unwanted application behaviour.

4. Conclusions

In conclusion, there are many advantages of a peer to peer service vs a traditional client-server architecture. A peer to peer service potentially allows contents to be downloaded faster and could free the network system of any possible bottlenecks. However, there are also tradeoffs to this methodology as downloading from sources that are not centralized could potentially compromise security as it is considerably harder to secure a network of computers versus one single point of entry. Through this project, methods of handling UDP and TCP sockets, techniques in thread handling, and peer inter-messaging processes were discovered and implemented. In modern times, peer to peer services are used to transfer media and for sharding large files to be downloaded across a network of computers, often proving to be faster than downloading through one single point (ie torrents).