Networks Final Project Proposal

Paul Bass and Hayley LeBlanc

April 17, 2019

1 Introduction

For our final project, we plan to develop a small peer-to-peer file-sharing program. Peer-to-peer file-sharing services allow a single client to download a file from multiple other hosts acting as servers concurrently, eliminating the need for a central servers to service all requests. Along with potentially improving download speeds on the client's end, peer-to-peer file-sharing services ensures that there is not a single point of failure on the server's end. If one host that is acting as a server fails or loses connection to the network during a download, the client can easily recognize this and download the rest of the file from another server.

2 System and Process Structure

This project will be implemented with peer-to-peer processes, where each process can act as both a client and a server. To take full advantage of the peer-to-peer file-sharing idea, we plan to build this project to work best with three or more hosts - one client and at least two servers from which the client downloads a file.

Our plan for the communication between the client and servers is as follows. The client will first determine which other hosts have the requested file, and it will send a request to some number of these hosts specifying (1) which file it wants, and (2) which part of the file it wants. This will allow the server hosts to be stateless, which is important since we want to be able to easily switch to a new server if one drops out. The servers will respond by sending the specified part of the file to the client. Once the client receives a chunk from each server, it will write them to a file on its end and send another round of requests, repeating this process until it has received the entire file. If a server does drop out at some point, the client will recognize this either by noticing a -1 return from a `send` operation, or by not receiving the expected response from the server within a timeout period. The client can then locate another server that has the desired file and re-request the missing chunk from that server.

3 Messages

Our file-sharing project will require a number of types of application-layer messages, which are described below. We will develop a protocol to ensure that the necessary information is contained within the messages and to distinguish between the different messages over a TCP stream. The protocol will likely be based on HTTP, but customized for our application.

- A message sent from client to server to request a chunk of a file from the server. This message will include information like the file name, the position in the file to start sending from, and potentially the amount of data to send (if this is not a pre-determined constant). Requesting small amounts of the file at a time will make it easy to parallelize the download across multiple servers and will also make it easy to replace a server that drops out during the download.

- A message sent from server to client containing a chunk of the file requested by the type of message described above. Along with containing data from the file, it will likely provide information like the amount of data actually sent (in case the server reaches the end of the file before reading the chunk size) and potentially other information about the server.
- A message broadcast from the client to all servers running the program to ask whether they have the requested file.
- A response to the above query sent by a server to let the client know if a server has the requested file or not.
- A set of error messages to prevent undefined behavior - for example, a message to let a client know that it has requested data beyond the bounds of a file or that it has sent a data request (not a query) for a file that does not exist or has restricted permissions on the receiving server.

4 Concurrency

There are a number of places in this project where concurrency must be addressed. Although each host can act as both a client and server, the two roles have different considerations when it comes to concurrency. We plan to primarily use `pthreads` to ensure that hosts can both listen, send, and receive concurrently

Each client will be receiving messages from multiple servers at the same time (or in quick succession), and it will need to correctly process and save the information received from each server. Clients also must be able to listen for requests for new files made by a human user while also communicating with servers. Servers must have the ability to send different files to clients simultaneously, and should always be listening for and be able to respond to new messages from clients.

Since this is a peer-to-peer system, any host should be able to act as a client and as a server simultaneously. Our main goals are handle the concurrency issues described above, but we also hope to develop our project so that a host can act as both a host and download a file from other servers while simultaneously sending a (different) file to someone else.