Lab 1: Peer-to-Peer File Sharing

CSE 514: Computer Networks
The Pennsylvania State University
Fall 2023
Due Date: October 12, 2023

1 Introduction

In this lab, you will design and implement a simple peer-to-peer (P2P) file sharing system. The goal of the assignment is to provide experience on implementing a P2P network.

2 System Description

The system will include multiple clients (also named peers) and one central server. A peer can join the P2P file sharing system by connecting to the server and providing a list of files it wants to share. The server shall keep a list of all the files shared on the network. The file being distributed is divided into chunks (e.g. a 10 MB file may be transmitted as ten 1 MB chunks). For each file, the server should keep track of the list of chunks each peer has. As a peer receives a new chunk of the file it becomes a source (of that chunk) for other peers. When a peer intends to download a file, it will initiate a direct connection to the relevant peers to download the file. A peer should be able to download different chunks of the file simultaneously from many peers.

3 Protocol Requirements

There are many ways to approach this project. At its core is a messaging system amongst peers and between peers and the server. You may design the protocol in any way you wish as long as it fulfills the system description and requirements.

Below is a minimal list of messages you can use. Feel free to add more messages if you like; however if you design your project carefully, these are the only messages you will need for a minimal yet successful design.

4 Basic Requirements

You can form a team and each team should consist of at most two people. Your project must fulfill the following requirements:

- **Multiple Connections:** Your peers and servers must be able to support multiple connections simultaneously. You cannot delay any message arbitrarily because of lack of parallelism. (Hint: use multithreading or select() or epoll().)

Peer Request	Server/Peer Reply
Register Request: Tells the server what files the peer wants to share with the network. Takes in the IP address and port for the endpoint to accept peer connections for download; the number of files to register; and for every file, a file name and its length.	**Register Reply:** For each file, it advises if the file registration was a success.
File List Request: Asks the server for the list of files.	**File List Reply:** Includes the number of files in the list; and for each file, a file name and a file length.
File Locations Request: Asks the server for the IP endpoints of the peers containing the requested file.	**File Locations Reply:** Includes number of endpoints; then for each endpoint, chunks of the file it has, an IP address and port.
Chunk Register Request: Tells the server when a peer receives a new chunk of the file and becomes a source (of that chunk) for other peers.	**Chunk Register Reply:** Advises if the chunk registration was a success.
File Chunk Request: Asks the peer to return the file chunk. Reads in a file name, chunk indicator.	**File Chunk Reply:** A stream of bytes representing the requested chunk (array of bytes).

- **Parallel Downloading:** Your system should maximize its download speed by receiving a file from multiple peers and assemble the file. For example, before downloading, the client can get the file size from a server and be able to divide the file into several chunks to be downloaded from multiple peers simultaneously.
- **Chunk Download Completion:** Upon finishing the download of a chunk, a peer must register that chunk with the server to become a source (of that chunk) for other peers.
- **Integrity Check:** To ensure integrity of downloaded file, a hash function should be used. You should test this by modifying a chunk at a peer, and other peers should discard it when they receive the modified chunk.
- **Chunk Selection:** When a peer downloads chunks from another peer, it should have an option of using "rarest first" to determine which chunk to download. In the demonstration, your program should be able to output which chunks of the file its peers have, and from which peer it downloads the chunks of the file.

5 Interface Requirements

The user should be able to specify which files to share as command line arguments. You may specify a directory or a list of files.
You must be able to ask for the file list and choose a file to download.
You must be able to view progress on active downloads.

6 Programming Language Requirements

You may implement this project in the language and operating system of your choice as long as you can provide a live demo. Most people will benefit from developing this project in C under a UNIX environment. Other choices include C++, C#, Python, or Java. Some platforms are easier than others so choose wisely.

7 What and How to submit

Your submission should include:

1. A detailed description of the system you built, along with protocol specifications.
2. A description of the structure of your program.
3. A description of which part works and which part doesn't.
4. A sample output of your program. Please notate the output so that it can be understood easily.
5. Well-commented source code.
6. A system description with makefile or compile command. Note, we only accept documents in pdf or txt format.

Then,

1. Compress all above materials in a tar file or zip file named lastname_firstname_1.zip (or .tar).
2. Submit it through Canvas.
3. Please prepare a demo for your project.

8 Resources

- You can use machines in your own lab or machines in the second floor of Westgate building such as W135 or W136. You can also "ssh" to the machines at other locations.
- You need to use socket programming to finish this lab. You can search for "socket programming" to be familiar with it.
- The following books might be helpful for this assignment.
  - "Unix network programming" by W. Richard Stevens
  - "Internetworking with TCP/IP" series by Comer.
- Compiling C programs under Solaris

```
gcc -lsocket -lnsl -o <output> <input.c>
```

- Compiling C programs under Linux

```
gcc -o <output> -lnsl -lsocket <input.c>
```

- You might need the following header in your C source file.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
```