Lab 1: Peer-to-Peer File System

CSE 514 – Computer Networking
Pennsylvania State University
Fall 2015
Yi Yang
yzy123@psu.edu
Due Date: Oct 15, 2015

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

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 and their locations (IP address and port). 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 a file simultaneously from many peers. Upon leaving the network, the peer must remove its files from the server's file list.

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.

Between Peer and Server

Peer Request	***Server Reply***
Register Request: Tells the server what files each peer wants to share with the network. Takes in the IP address (uint32) and port (uint16) for the endpoint to accept peer connections for download; the number of files to register (uint16); and for every file, a file name (string) and its length (uint32).	**Register Reply:** For each file, it advises if the file registration was a success (Boolean).
File List Request: Asks the server for the file list.	**File List Reply:** Includes the number of files in the file list (uint16); and for each file, a file name (string) and a file length (uint32).

File Locations Request: Asks the server for the IP endpoints of the peers containing the requested file (string).

File Locations Reply: Includes the file size of the requested file (uint32); number of endpoints (uint16); then for every endpoint, an IP address (uint32) and port (uint16).

Leave Request: Tells the server remove this peer and all its files from the network.

Leave Reply: Advises if the peer was removed successfully (Boolean).

Basic Requirements

All projects 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 fork().)

Parallel Downloading: Your system should be able to download 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 blocks to be downloaded from multiple peers simultaneously. To ensure integrity of downloaded file, hash function should be used (see bitTorrent). In the demonstration, you need to divide a file into at least two segments and download them from different peers.

Failure Tolerance Mechanism: Your programs must not crash if a peer or the server unexpectedly fails or leaves the network. Malformed messages must not crash your programs, either. Make sure that your program removes the files registered on the server when a peer leaves the network.

Download Completion: Upon finishing a download, a peer must register that file with the server so that other peers can use it.

Group Requirements

If you wish, you may form a group with at most one other person. The following requirements are optional for individual work but required for group work:

Download Optimization: Your peers must be able to adapt the downloading plan on the fly; they should be able to add/remove peers as they become available and exploit faster peers. (Hint: You can test this by making certain peers purposely slower by specifying an upload delay command line argument).

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.

Other Considerations

These are some things you may want to keep in mind while implementing this project. While you don't need to *necessarily* address these issues, they may make your project easier to implement and are good experience anyway.

What will you do when peers try to register a file with the same name but different file size or contents?

How will you structure your downloading plan amongst peers? Should you split the entire file upfront amongst connected peers or should you download in smaller file blocks? How large would those file blocks be? Should you use a queue to schedule these file blocks amongst peer connections?

What will you do if all the peers leave in the middle of a download? Will you error the file or will you wait until peers become available again? Would this naturally lead to a pause/resume feature for these situations?

Implementation Requirements

You may implement this project in the language and operating system of your choice (but no scripting languages) as long as you can provide a live demonstration. Most people will benefit from developing this project in C under a UNIX environment. Other choices include C++, C#, or Java. Some platforms are easier than others so choose wisely. Regardless, your code must use the local TCP libraries directly and should not use any additional layers (HTTP or XML services, for instance).

You can use machines in your own lab or machines in the second floor of IST building such as 218 IST or 222 IST. You can also "ssh" to the machines at other locations.

UNIX and C

The following books might be helpful for this assignment.

"Unix network programming" by W. Richard Stevens

"Internetworking with TCP/IP" series by Comer.

Compiling under Solaris:

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

Compiling under Linux:

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

You might need the following header in your source file:

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

C# and Windows

You might need the following assemblies:

```
using System.Net;
using System.Net.Sockets;
using System.Threading;
```

Deliverables

Your submission should include:

1. A detailed description of the system you built, along with protocol specifications.
2. A description of what works and doesn't.
3. A description of how you handle fault tolerance.
4. A description of the structure your source code.
5. Well-commented source code.
6. A makefile, Microsoft Visual Studio project, or other build instructions.
7. Sample output from your program. Please notate the output so it can be understood easily.

Your descriptions and sample output must be in the same document and must be in Microsoft Word, PDF, or plain text format. To submit the lab,

1. Place all your files in a single directory named lastname_firstname
2. Compress the directory into a file lastname_firstname.zip (or .tar.gz)
3. Submit it in an attachment to Yi Yang (yzy123@psu.edu). The email subject should be "CSE 514 – Lab 1 Submission".
4. Please prepare a demo for your project.