CS 5600 – Computer Networks
Course project description

Project Specification:
In this project, you will build a peer-to-peer file sharing protocol based on the similar lines of the well-known bit torrent protocol. At the core of this protocol is a *tracker server* that keeps track of which peers are sharing and what files are being shared in the network. The peer program contacts the tracker server to create tracker entry and update its sharing status. Other peers can download the shared file. A peer can download a shared file from more than one peer sharing it. Only the peers (known as seeds) that have full files to share can send *createtracker* message to the tracker server. The peer program periodically sends *updatetracker* messages to the tracker server. It can contact the tracker server and request the list of tracker files maintained by it. Additionally it can request the tracker server for a particular tracker file so that it can initiate the file transfer process. As soon as the tracker file is received, the requesting peer redirects its connection to the selected peers and the download of the file starts automatically.

Implementation Issues:
Development of the entire project consists of 3 major development steps:
1. Peer program
2. Tracker server
3. Protocol between the tracker server and the peers

Tracker File Format:
Tracker file can be created by a peer (seed) who has full file to share. The tracker file name will be same as the original shared file name. The information contained in the tracker file will be stored as:

Filename: <file-name>
Filesize: <size of the shared file in bytes>
Description: <short description about the file – optional>
MD5: <md5 checksum of the file>
all comments must begin with # and must be ignored by the file parser
following the above fields about file to be shared will be list of peers sharing this file
<ip address of the peer>:<port number>:<start byte>:<end byte>:<time stamp>
...
<ip address of the peer>:<port number>:<start byte>:<end byte>:<time stamp>

For example suppose peer A wants to share a movie file called movie1.avi whose file size is 109283519 bytes. Further say the peer's server thread is listening for other peer's connection at port number 2097 and additionally peer's own IP address is 202.141.60.10, it will create a tracker entry at the tracker server using the protocol message which will be described later. The tracker file that the tracker server will create will look like as shown below:

Filename: movie1.avi
Filesize: 109283519
Description: Ghost and the Darkness, DVD Rip
MD5: c68c2ee8bfca4e898b396e7a935a1d92
#list of peers follows next
202.141.60.10:2097:0:109283519:1174259923
10.126.155.17:20182:1500:123821:1174263923

The above details will be stored at the tracker server as file movie1.avi.track

Peer Program:
Peer program has the responsibility of creating multiple threads so as to communicate with other peers as well as the tracker server. It follows the following strategy for downloading a file:

1) Dividing file into segments: The downloading file is divided into many segments, and the size of each segment could be defined in any number in advance.
2) Segment selection: The to-be-downloaded segment(s) is (are) chosen sequentially.
3) Peer selection: The peer which has the newest timestamp is selected to be connected to download the corresponding segment.
4) Update: Only after downloading a complete segment does a peer update its record file and sends information to tell the tracker server that it has this part of the file.

Protocol Description:
P2P peer program must handle 'createtracker' and 'updatetracker' commands in order for the peer to be able to create / update tracker file at the tracker server. The protocol format of the messages between the peer program and the tracker server will be:

createtracker: message from peer to the tracker server
<createtracker filename filesize description md5 ip-address port-number>\n

createtracker: message from tracker server to the peer program
if command successful: <createtracker succ>\n
if command unsuccessful: <createtracker fail>\n
if tracker file already exists: <createtracker ferr>\n

Consider the same example described above, the protocol message for the message from the peer to the tracker server will look like:
<createtracker movie1.avi 109283519 Ghost_and_the_Darkness,_DVD_Rip
c68c2ee8bfca4e898b396e7a935a1d92 202.141.60.10 2097>\n

P2P peer program must also periodically update the tracker server with the fresh description of the files it is currently sharing. This is required because the file share might change at the peer as it is still in the process of downloading the file or it might decide to no longer share a particular file anymore. You must make the time interval between successive tracker refresh by the peer part of its configuration file. Make this parameter

15 minutes by default in the peer configuration file.

updatetracker: message from peer to the tracker server
<updatetracker filename start_bytes end_bytes ip-address port-number>\n

updatetracker: message from tracker server to the peer program
if tracker file does not exist: <updatetracker filename ferr>\n
if tracker file update successful: <updatetracker filename succ>\n
any other error / unable to update tracker file: <updatetracker filename fail>\n

You can make the peer program automatically send updatetracker messages to the tracker server for each file in the peer shared directory every 'n' seconds where 'n' is the refresh frequency time in the configuration file.

P2P peer program must be able to contact the tracker server and request the list of tracker files maintained by it using LIST command.

LIST – This command is sent by a connected peer to the tracker server to send over to the requesting peer the list of (tracker) files in the shared directory at the server. The format of the incoming message from the connected peer will be
<REQ LIST>\n

In reply to the LIST request the server reply message structure must be:
<REP LIST X>\n
<1 file1name file1size file1MD5>\n
<2 file2name file2size file2MD5>\n
...
<x fileXname fileXsize fileXMD5>\n
<REP LIST END>\n

Additionally the P2P peer must be able to request the tracker server for a particular tracker file so that it can initiate the file transfer process. Protocol for the message sent by P2P peer to the tracker server in order to request a particular tracker:

<GET filename.track >\n

The server's response to the GET command must be:
<REP GET BEGIN>\n
<tracker_file_content >\n
<REP GET END FileMD5>\n

Once the P2P peer receives the reply for its tracker file request (GET), it must compare the MD5 checksum of the received data with the one contained as part of the protocol message. If MD5 matches, the tracker file data is assumed correct. The peer must save the tracker file into its local cache.

The peer must be able to intelligently create peer requests using GET in order to get maximum download speed. Maximum data chunk size must be set to 1024 bytes. P2P peers' server thread handling the GET request by other peers must check for the chunk size requested and must enforce 1024 bytes upper limit strictly. In case the server thread at the peer receives a GET request for file chunk size greater than 1024 bytes, it must respond back with an error message <GET invalid>\n

Client/Peer must be able to create multiple threads, each thread requesting mutually exclusive file chunks from different peers, merging different chunks if possible after each thread terminates. This process of downloading the file must automatically start once the tracker file has been received successfully. The peer must be aware that the received tracker data might not be up to date as the tracker update period interval by peers may be large, so it should be prepared for failed TCP connection attempts made on some of the listed peers in the tracker file and deal with the situation intelligently. The peer must also be able to handle its own failure appropriately. That is peer program when executed must check into its local tracker cache and the shared file storage location to see if there are still any incomplete files and must try to download the remaining bytes and not start the whole process over again. Once the file is successfully downloaded in its entirety, the peer must delete the corresponding tracker file stored in its local cache. The peer must contact the tracker server periodically [period of tracker update by peer should be read from the configuration file]. The client must check for incomplete file chunks in its local storage after each period elapses and should ask the tracker server for the latest tracker for only the files which are still incomplete.

Tracker Server:

This is a multi-threaded centralized server whose primary job is to maintain a list of peers sharing either partial or complete file chunks for each shared file in the P2P network. Each new peer request connection should immediately be handed over to a worker thread that shall handle the request from that peer. Each worker thread serves only one peer at a time and terminates the connection as soon as the protocol reply message has been sent to the requesting peer. The commands received by the tracker server are: LIST, GET, createtracker, and updatetracker.

Upon receiving the commands, the tracker server behaves in the following ways:

1. LIST: sends the list of tracker files (with added information).
2. GET: sends the tracker file being requested.
3. createtracker: creates a tracker file with received information and time stamp, if the same tracker file is not already created, and sends error message, otherwise.
4. updatetracker: if no such file exists it responds back by an error message to the peer, closes the TCP connection and terminates the handler thread. If such tracker file exists, then it creates a new entry if the peer is new (the time stamp for this new entry will the system's current time stamp) and updates the information if the peer is already added (its time stamp must be updated to the current system time stamp.). It also removes the entry of the dead peers. A peer is considered dead if its update time interval elapses.

Project Guidelines

Failure to follow the required guidelines will penalize every member of a team. There will be at least 10% deduction in your project grade if a required guideline is not followed.

Implementation:

1. Every group member must contribute (almost) equally.
2. You are recommended to implement in C as it will make your implementation a lot easier. However, you are allowed to use any language you prefer and language choosing will have no impact on grading.
3. Add sufficient comments in your code.
4. You must handle all errors.
5. Keep all necessary information (such as port number, IP address) in configuration file. Each machine must maintain its own configuration file.
6. The underlying TCP protocol messages format are to be strictly adhered to and file naming conventions and tracker file content formats must also be strictly adhered to.
7. Program must display on screen necessary events (e.g. "file x requested from y, z", " File x download complete".. so on).
8. Your project submission must include a valid makefile along with other documentation files and source code. You should not include any compiled object files with your submission.
9. Mention any external codes / libraries / packages you used in your code along with links to those resources.
10. Please clearly mention the role of each individual team member in developing this project in your report.

Mid-term demo: on October 22
Each group is expected to show multithreaded implementation as well as file transfer between two machines.

Mid-term report:
Write a one-page project report stating what you have done so far, what are left, what was the role of each member. Group member names must be sorted alphabetically on last name (Therefore, no worries if you are not the first author!).

Final demo: on Dec 1 and 3.

Final report: due by 11:59pm Dec 4
Final report must not exceed 5 pages 10 point font size on ACM/IEEE or any format you choose as long as the side margins do not exceed 1 inch. **The report must be PDF**. The report must include at least the following:

1. Member names sorted by last names.
2. Each member's role.

3. Citations of the sources of any code, library, package used.
4. Code design.
5. Installation guide and running guide.

Final submission:
Your project submission must include a valid makefile along with other documentation files and source code. You must not include any compiled object files with your submission. Add all source files, a makefile, and the report (in PDF only) in one folder. Name the folder as X_Y_Z where X, Y, and Z are the lastnames of the group members. Zip the folder and email to the TA by 11:59pm ~~Dec 5.~~ Dec 4.

Grading:
1. 25% in midterm demo and report
1. 5% of total grade will be assigned for good coding style including adequate comments, indentation etc.
2. 5% of total grade will be assigned for proper documentation including installation guidelines, code design document.
3. 5% of total grade will be assigned for clear / clean user interface design that is displaying only necessary output on the screen / terminal
4. Rest 60% will be reserved for correct project implementation.

How To Run the Tracker Server

Use makefile to compile the tracker server. The following command will compile it:

`make –f makefile`
This will generate the executable file ***tracker***. The following command will run the tracker server program: ***`./tracker`***

The server, at this stage will start listening incoming request. All the tracker files will be created and saved in the folder named ***torrents*** in the root directory. This is the shared directory of the tracker server. This shared directory and port number are read from the configuration file ***`sconfig`***.

How To Run the Peer Program:
It will suffice if a group uses one machine as tracker server and one machine for peers where the latter will simulate multiple peers. To ensure consistency among all groups, each group will need to simulate exactly 3 peers, named peer1, peer2, peer3. The easiest way to do this is to compile the same peer program, say **peer.c**, and make 3 executables in three different folders. Specifically, create a makefile for the peer that will compile **peer.c** and output the executable files to folders peer1, peer2, and peer3 (these folders mimic separate peers even working with a single machine).

Config files formats:
Use two configurations file for each peer as follows.
clientThreadConfig.cfg: First 2 lines are port no and IP address of the tracker server, and last line is the periodic updatetracker interval in seconds. (**This configurations, especially the Tracker Server address, must be set before the programs are started**.)

serverThreadConfig.cfg: First line is the port no to which the peer listens, i.e. for incoming peer connection requests, and last line is the name of the shared folder. Note that, each peer obtains its own IP automatically; so, that info doesn't exist in files. Once compiled (make), you need to enter **`./peer`** to run each peer.

Message Format:
While the following commands will need to be automatically sent from the peer, you are also required to have the option for manual testing of the following commands. We may try to give the following commands manually and test whether your program works:

LIST command format: `REQ LIST`

GET command format: `GET filename.track`

createtracker: `createtracker filename filesize description md5 ip-address port-number`

updatetracker: `updatetracker filename start_byte end_byte ip-address port-number`

Project midterm demo guidelines

Requirement: *You have to show that you can establish connection between two machines and request and download files.*

Platform: Linux Ubuntu 64-bit

Manually test the following commands:

The protocol format of the messages between the peer program and the tracker server will be:

1. createtracker: message from peer to the tracker server
<createtracker filename filesize description md5 ip-address port-number>\n

createtracker: message from tracker server to the peer program
if command successful: <createtracker succ>\n
if command unsuccessful: <createtracker fail>\n
if tracker file already exists: <createtracker ferr>\n

2. updatetracker: message from peer to the tracker server
<updatetracker filename start_bytes end_bytes ip-address port-number>\n

updatetracker: message from tracker server to the peer program
if tracker file does not exist: <updatetracker filename ferr>\n
if tracker file update successful: <updatetracker filename succ>\n
any other error / unable to update tracker file: <updatetracker filename fail>\n

3. LIST – This command is sent by a connected peer to the tracker server to send over to the requesting peer the list of (tracker) files in the shared directory at the server. The format of the incoming message from the connected peer will be
<REQ LIST>\n

In reply to the LIST request the server reply message structure must be:
<REP LIST X>\n
<1 file1name file1size file1MD5>\n
<2 file2name file2size file2MD5>\n
...
<x fileXname fileXsize fileXMD5>\n
<REP LIST END>\n

4. GET: For mid-term demo, you can download the requested file from either the tracker server or from another machine (make sure the file is already in that machine).

<GET filename.track >\n

The server's response to the GET command must be:
<REP GET BEGIN>\n
<tracker_file_content >\n
<REP GET END FileMD5>\n

Report submission:
Write a one-page project report stating what you have done so far, what are left, what was the role of each member. Group member names must be sorted alphabetically on last name (Therefore, no worries if you are not the first author!). Submit your report on 10/22 Thursday in class.

Forming Group:

This is a group project. Each group should have 3 members (there may be few exceptions). **One member from each group must email all group member names to the TA by the end of ~~Friday 09/11~~** Tuesday 09/15.

Tutorial on socket programming:

For socket programming, you may use any language and platform. But using C on Linux will make your implementation much easier. *"Beej's Guide to Network Programming Using Internet Sockets"* is a good tutorial to learn socket programming.

Alternative Project

Each group can select its own project upon the instructor's approval. The project must be a networking project and must be implemented on real hardware and demonstrated before the entire class. Simulations results will not be accepted.

Each group that selects a different project must follow the same guidelines for project report, demonstration, and milestones. In addition, each of these groups must send their project proposal through an email to the instructor detailing the project. **The deadline for sending project proposal is Friday 9/11.**