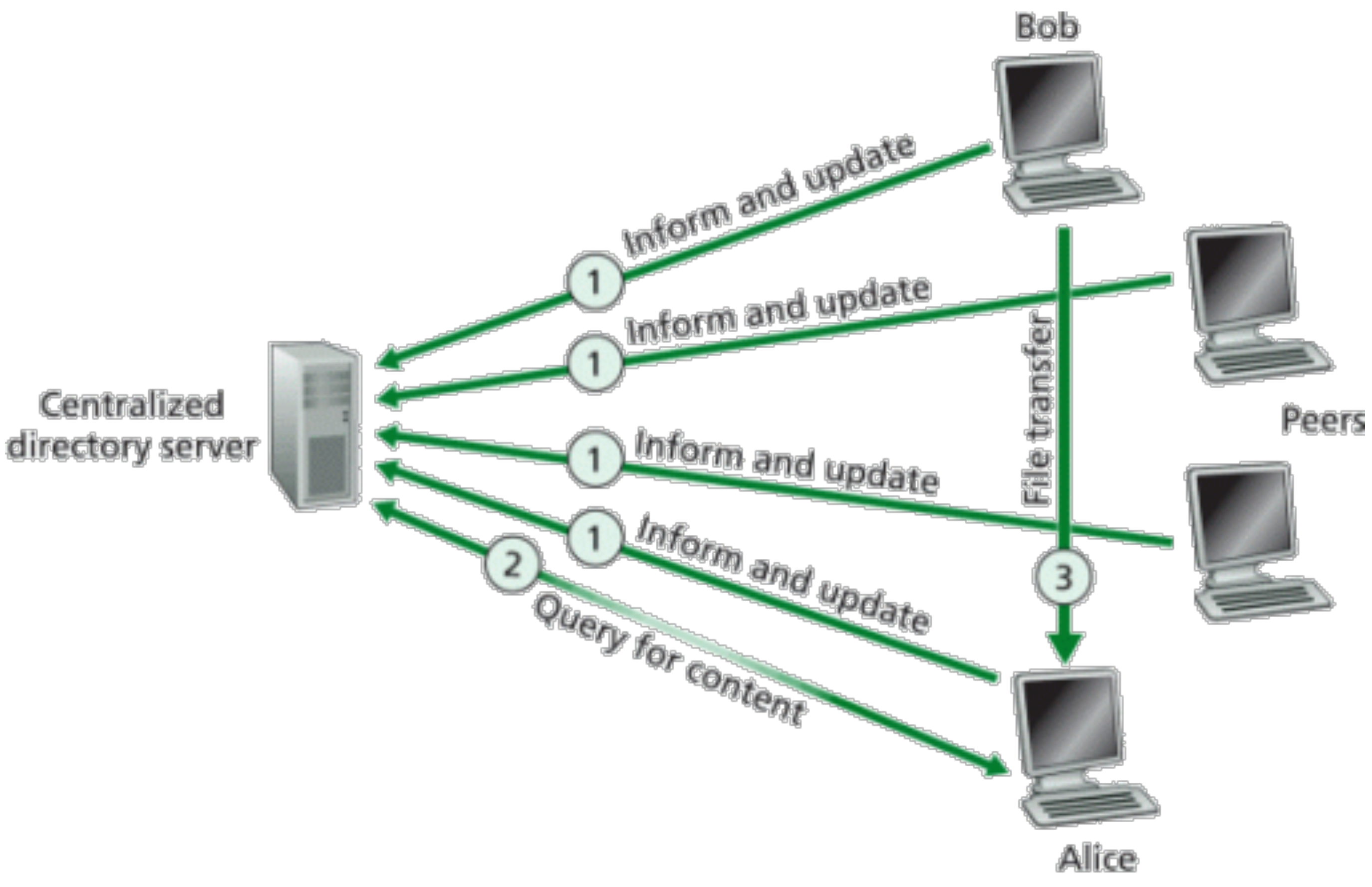
CS3103 P2P File Transfer Project

Learning Objectives

This project provides an opportunity to learn different p2p architectures, NAT traversal techniques, p2p system design challenges and implementation techniques. You will design the system architecture and all relevant protocols.

Project Description:

The application will consist of two components: (a) a server discovery component and (b) a file transfer component. You must use basic sockets to implement client/server communication. You can implement using either C/++ or Java or Python programming language.



MISSION 1: Basic P2P File Transfer [70 %]

We assume that this system can be used to share chunks of a file. Each peer is uniquely identified by the name of its host computer; each host has a unique IP address (in the same Network, no need to cross NAT). The system consists of a centralized directory server and a number of peers; each peer combines a functionality of a P2P client and a transient P2P server. In our system, the P2P client can acts as the client to the directory server to obtain information about available files; it also acts as the client of a P2P transient server in order to obtain desired files. Therefore, this project consists of two major parts: implementation of the communication protocol between the directory server and a P2P client, and implementation of the communication protocol between a P2P client and a transient P2P server.

a) Directory server (Tracker) and its interaction with P2P client

The centralized directory server maintains a directory of chunks (of different files) which users are willing to share. It contains entries which list *file name, chunk number, and the host name* where the chunk resides. When a user, say Bob, wants to advertise a chunk for sharing, he sends a complete directory entry to the directory server using an "Inform and update" message. This message is sent using UDP (TCP if you prefer!). Each "inform and update" will inform the server about set of chunks available at the client for sharing. The directory server acknowledges the message. Design your own message format (eg. like HTTP header and body) for this communication.

When a client wants to exit from this application, it sends an "exit" message upon receiving which the server deletes the entries associated with this client. When a P2P client wants to get some content from its peers, it sends a "query for content" message to the directory server specifying the *file name, chunk number or asking for a directory listing from the directory server*. Design your own format for all messages sent to the directory server. The directory server should be multi-threaded and should handle concurrent writings to the media file directory.

b) P2P client and its interaction with P2P server

A P2P client is a client of both the directory server and the P2P server. The P2P client provides a text-based user interface for a user to,

1. Query the centralised server for list of files available.
2. Query centralised server for a specific file.
3. Download a file by specifying the filename.
4. Inform availability of a new file (file name) and its chunks (chunk numbers).

No particular format is enforced on how to maintain the database of chunk entries at the directory server. Similarly, no particular format is enforced for the query results of a directory listing request. However, the choices of these formats including the message formats of the messages to/from directory server must be **documented in the final protocol design report**. The report should also include description on how to use the user interface to issue commands and how a P2P server selected in case there are multiple P2P server with the same chunk.

A P2P server will be able to handle multiple simultaneous requests in parallel. This means that the P2P server must be multi-threaded. In the main thread, the server listens to a fixed port. When it receives a TCP connection request, it creates a new TCP socket and services the request in a separate thread.

You can use different port numbers for the directory server and the P2P servers.

Note:

You do not need to worry about 'performance' improvement and free riders (no need to implement tit-for-tat). You just need to handle chunk system (dividing a file into chunks), uploading and downloading chunks. If chunks of a file is available in multiple peers, the client can select any one of them in Random or with some policies to balance the load.

MISSION 2: Handling Nat [30 %]

Assume your P2P server host uses private IP address and it is behind the NAT (non-symmetric version is sufficient). You can use any technique (eg. STUN/TURN server) to find P2P server's public IP address and port number (that is the NAT entry) and inform the directory server periodically. The directory server's data base entry now should include the *public IP address* and *port number* in addition to *file name, chunk number, and the host name.*

Clients should be able to connect to this public IP address (NAT router's IP address) to get access to the P2P server.

Note: In addition to VMs in SoC, you may need some external VMs for testing. Some vendors offer free VMs (eg. https://www.heroku.com/pricing) and some offer for a price as low as $5 per month (eg. https://www.digitalocean.com/pricing/).

MISSION 3: Distributed Directory [Bonus]

Devise a mechanism (distributed hash table) to distribute the directory information of centralized server to multiple P2P servers. Design a protocol for the clients to find the availability of a chunk by minimum number of queries to the peers.

Administrative details

1. **First submission:** Protocol design doc (<= 2 pages, excluding diagrams) - **by 17th Oct. [in IVLE - 'Project Stage 1 (Protocol Design) Submission' folder.** 1 copy per group, any member of the group can upload**]**
2. **Second submission:** Project prototype demo (progress demo) - demo the codes that you have implemented by executing the code in your machine. Upload short video clip (screen capture of your demo) **by 26th Oct. [in IVLE - 'Project Stage 2 (Prototype Demo) Submission' folder.]**
3. **Final submission:** Submit the final version of your source code, instructions for running, and updated protocol-design combined into a single RAR file to IVLE **by 13th Nov. [in IVLE - 'Project Stage 3 (Final Code & Docs) Submission' folder.]**
   PROJECT FINAL DEMO TO CLASS: 15th (Thu) and 16th (Fri) Nov - during the lab sessions. **Venue: Lab**
4. Submit intra group (within group) peer evaluation by 19th Nov to "IVLE->Intra-Peer Eval Submission" Folder. Evaluation form is available in IVLE.

----- THE END -----