ASSIGNMENT 1

DEVELOP A NETWORK APPLICATION

COURSE: COMPUTER NETWORKS, SEMESTER 1, 2024-2025

OBJECTIVES

Build a **Simple Torrent-like Application (STA)** with the protocols defined by each group, using the TCP/IP protocol stack and must support **multi-direction data transfering** (**MDDT)**.

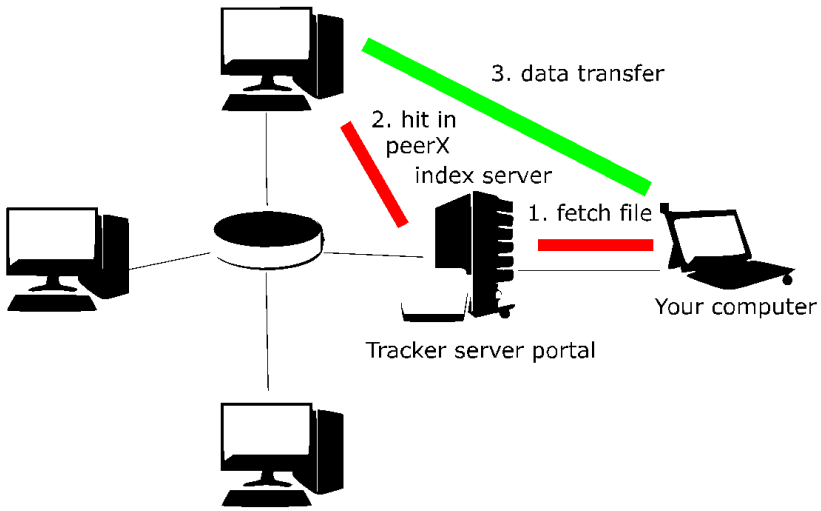
APPLICATION DESCRIPTION

Application overview

- The application includes the two types of hosts: tracker and node.
- A centralized tracker keeps track of multiple nodes and stores what pieces of files.
- Through tracker protocol, a node informs the server as to what files are contained in its local repository but does not actually transmit file data to the server.
- When a node requires a file that does not belong to its repository, a request is sent to the tracker.
- **MDDT**: The client can download multiple files from multiple source nodes at once, simultaneously.

This requires the node code to be a multithreaded implementation.

Figure 1. Illustration of file-sharing system.



The simple Like-torrent essential components

Before starting with the network transmission, you need to know something about all these small crucial components:

- **Magnet text:** is simple text that contains all of the necessary information to metainfo file on the tracker. For minimum requirement is a hash code point to the metainfo file/information stored on your centralized tracker portal (aka your-tracker-portal.local).
- **Metainfo File:**(also known as .torrent file) holds all the details about your torrent, including where the tracker address (IP address) is, what is the piece length, and piece-count. It can be considered as the info where magnet text points to and is combined with the current list of trackers and nodes.
- **Pieces:** are the unit of file' parts. Pieces are equally-sized parts of the download. The usual size of the piece is around 512KB. Piece info is listed in Metainfo File
- **Files:** are also specified in the Metainfo File. File includes multiple pieces. There can be more than one file in a torrent. This implies that you may need to map the piece address space to the file address space if you have N parts and M files.
    - *Simple math mistakes are very likely possible here, but you MUST carry out the entire mapping between File and the associated pieces by hand and by yourself definition for later referring.*
    - *For starting point, you can implement one file per torrent as a minimum score but stopping at this stage could limit your scoring evaluation.*

Tracker HTTP protocol

Tracker request parameters: at startup, you are required to contact the tracker server (the global centralized tracker portal), and submit all the appropriate support info fields.

- *You should include the compacted flag(s) or metainfo in your request, a bare minimum requirement is a magnet text of your torrent that is able to parse (to locate to a Metainfo File placed on the server).*
- *We can refer the sample netapp example with two phases: the client-server paradigm and the peer-to-peer paradigm*
- Lastly, you have *to* submit started, stopped and completed requests to the tracker server when appropriate. When you send the started request, you need to include how many bytes you have downloaded already (at the piece level detail).

Tracker response: we determine a limited subset of the specification for bittorrent (in https://wiki.theory.org/BitTorrentSpecification).

- The tracker is initially defined as centralized and single tracker, but there is no limit to upgrading it to multiple tracker with a reference theory DNS multiple servers using iterative or recursive query.

The tracker responds with "text/plain" document consisting of a dictionary with the following keys:
Failure reason: If present, then no other keys may be present. The value is a human-readable error message as to why the request failed (string).
Warning message: (new, optional) Similar to failure reason, but the response still gets processed normally. The warning message is shown just like an error.
Tracker id: A string that the client should send back on its next announcements. If absent and a previous announcement sent a tracker id, do not discard the old value; keep using it.

Peers: (dictionary model) the value is a list of dictionaries, each with the following keys:

- peer id: peer's self-selected ID, as described above for the tracker request (string)
- ip: peer's IP address either IPv6 (hexed) or IPv4 (dotted quad) or DNS name (string)
- port: peer's port number (integer)

PEER- Downloading

After you get the list of peers from the Tracker, you now must connect to as many of them as you can and start downloading from them.

- *It is mandatory to support **MDDT**.*
- Your algorithm for choosing which blocks to request from which peer is completely up to you, with the minimum requirements that you don't send a request for a piece the peer does not have, and *you don't send a request for a piece you already have*.

For advanced development:

To maximally utilize network bandwidth, you want to have a **request queue**, which keeps a list of all blocks you have sent a request message to the peer for.

Since you will have multiple peers for the same torrent, you will need some way to keep **current status progress** of which blocks have been requested to all the peers, so you don't send requests for the same block to multiple peers and it supports recovery from disconnect on-going file transfer.

PEER- Uploading

It wouldn't be possible to download files with your client without seeders. Therefore, your application has to begin seeding the file to other peers who are also interested in downloading it after you download it.

For advanced development:

The "tit-for-tat" theory underlies peer-to-peer file sharing, implying that the more you share, the more files will be shared with you. This project is a chance to apply your learned theory.

The currently deployed peer selecting algorithm prevents free-riders by only changing peer selecting decisions once every ten seconds.

User Interface

This part of the assignment allows for a large room for creativity: you can utilize and implement any style of client you want, as long as it satisfies the minimum requirements of downloading and seeding a single torrent to multiple peers, and provides us ability to see all the upload/download statistics.

In general, http://www.transmissionbt.com/has a complete description. You can support some fundamental parameters and options of the transmission-client command lines. We keep an in-depth description for your comprehensive reference in future research if needed.

- **transmission-cli** - a bittorrent clienthttps://linux.die.net/man/1/transmission-cli
  - E.g: transmission-cli <<*torrent-file*>>
- transmission-create: https://linux.die.net/man/1/transmission-create
- transmission-daemon: https://linux.die.net/man/1/transmission-daemon
- transmission-edit: https://linux.die.net/man/1/transmission-edit
- transmission-remote: https://linux.die.net/man/1/transmission-remote

- transmission-show: https://linux.die.net/man/1/transmission-show

You could even develop a GUI if you have a lot of extra time. Although you probably won't get extra credit for this, you might earn some Github stars instead.

Extra credit:

You may choose to implement any or all of the following items (or something event not in this list but it is relevant to bittorrent and networking)
Distributed hash table (DHT): transmit your torrent file without the centralized tracker. Typically, this is a better source to find more peers here than by using the torrent's centralized tracker.
Simultaneous torrents: require your client show how to download and upload several torrents simultaneously (at the same time), as well as access the statistics through your user interface.
Tracker scrape: https://en.wikipedia.org/wiki/Tracker_scrape is the exchange to obtain the metainfo on behalf of the client. It should be mentioned that a peer's involvement in a data transmission is unaffected by scrape exchanges.
Download/Seeding strategies: a simple download strategy is required for your client, but if you choose to apply something fancier (like Super Seeding, Rarest-First, End Game), you need to document it and provide supplemental materials.
Optimizing peer selection: a simple peer selection strategy is required for your client, meaning you can just response to all your peers. If you want to use the wiki's actual standards (of only 4+1 peers at a time), please make sure you document it

Grading

The main requirements for this assignment are:

- Download a multi-file torrent from multiple peers concurrently
- Upload a multi-file torrent to multiple peers concurrently
- Show useful statistics of download/upload statistics, that allow us to inspect peer information, currently used torrents.

You grade breakdown will be as follows:
Tracker Protocol - 15% successfully parse the metainfo, send a request to the tracker, obtain its response with the list of corresponding peers, and parse the list of peers into useful 'ip' and 'port' pairs.
Torrent Download - 30% establish connection to several peers, download the torrent from all of them simultaneously, and appropriately use the mapping to determine which blocks to request. You must not send requests for pieces you have stored on disk and you must handle multi-file downloads correctly.
Torrent Upload - 15% allow several peers to connect to you so that you can serve multiple request streams simultaneously.
Multiple host deployment - 20% this course is the network connect among computer hosts, therefore the deployment of demo with more than 3-4 host is mandatory (virtual machine technique is ready for implementing this requirement)
MDDT feature - 10%
Advance features - 10%

WORKING PLAN

- Work in a team.
- Each group has 2-3 or up to 4 members.

Phase 1 (First 2 weeks):

- Define specific functions of the file-sharing application
- Define the communication protocols used for each function

Phase 2 (Next 2 weeks)

- Implement and refine the application according to the functions and protocols defined in Phase 1

RESULTS

Phase 1 (softcopy):

- Submit the report file of the first phase to BKeL in the predefined section as announced on BKeL.
- File format (pdf): **ASS1_P1_<<Group_Name>>.pdf**

This report includes:

- Define and describe the functions of the application
- Define the protocols used for each function

Phase 2 (hard and softcopy):

- Submit the report file of the assignment to BKeL
- File format (.rar): ASS1_**<<Group_name>>**.rar

This report includes:

- Phase 1 content
- Describe each specific function of the application
- Detailed application design (architecture, class diagrams, main classes ...)
- Validation (sanity test) and evaluation of actual result (performance)
- Extension functions of the system in addition to the requirements specified in section 2
- Participants' roles and responsibilities
- Manual document
- Source code (softcopy)
- Application file (softcopy) compiled from source code

EVALUATION

Assignment 1 is worth 15% of the course grade.

DEADLINE FOR SUBMISSION

4 weeks from the assignment announcement.