CSCI 4430S - Assignment 1 (17.5% of course grade)

A Partial Implementation of BitTorrent.

Individual assignment - Released on 2012 Feb 20.

Abstract

BitTorrent (BT for short) has been the most successful and the most popular peer-to-peer (P2P for short) software. In this assignment, we are going to realize some feature of this software.

1 System Components and Protocols

A typical deployment scenario of a BitTorrent system is shown in Figure 1. We first explain the role of the following components play in this assignment.

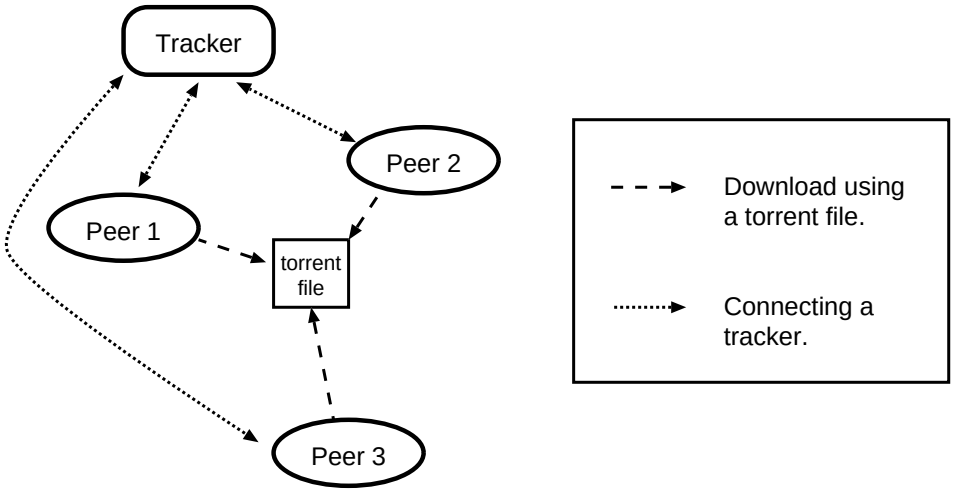


Figure 1: The components for this assignment and the possible connections among them.

For quick reference, we first list the assumptions below. We will echo the assumptions later.

- A tracker is always online and there are at most one tracker for this assignment.
- A peer in the peer list is guaranteed to be online.
- There are at most 5 peers in the system.

- A peer should handle at most 1 torrent file at the same time.
- Each torrent file contains one file only.
- A peer does not guarantee to be able to finish any downloads.
- A peer may quit before the download has been finished.

1.1 Components

1.1.1 Torrent file and torrent generator

We will provide you a torrent generator and that generator will produce torrent files according to a common format.

- **Role**. A torrent file tells a peer how to download shared files. In this assignment, we assume that a peer will handle at most one torrent at a time.
- **File content**. For each torrent, we assume that it only specifies **one file**. In later text, we will use the phrases "*downloading a file*" and "*downloading a torrent*" interchangeably. For each torrent file, it stores the following items:

Item	**Length**	**Description**
Torrent ID	4 bytes	It is a randomly-generated number of 4 bytes long. It is for the tracker and the peer to distinguish if a unique torrent is presented.
Tracker location	6 bytes	The first 4 bytes represent the little-endian representation of the IP address of the tracker. The next 2 byte represent the little-endian representation of the port number of the tracker.
File name	At least 5 bytes	The first 4 bytes represent the little-endian representation of the length of the file name. Let the length of the file name be n. Then, the next n bytes in the torrent file represents the file name in terms of an ASCII-character string. Note that the length of the file name could never be 0.
File size	4 bytes	The last 4 bytes of the torrent file store the little-endian representation of the size of the to-be-downloaded file.

Unlike the real BT application, the torrent file in this assignment does not contain any checksums concerning the downloading file.

1.1.2 Tracker

We will provide you the binary of a tracker. The following is the description about a tracker.

- **Role**. A tracker is a server process and opens a listening port. You can choose your own port number when the tracker starts. It allows multiple peers to connect at the same time.
- **Internal status**. A tracker stores the list of entries about which peer is downloading which torrent.
- **UI**. The tracker provides an interface for the user to invoke the following commands:

Command	**Description**
`list`	Print the internal status of tracker.
`exit`	Terminate all connections and terminate the tracker process.

1.1.3 Peer

You are required to implement different versions of peers.

- **Role**. A peer is a node in a P2P model. Hence, it should have *one listening port* opened and welcome incoming connections. The listening port number is not necessarily a pre-defined one. The system call `getsockname()` should be helpful in reading the port number of a socket. Then, the peer can tell the tracker its listening port number and its IP address.

  Meanwhile, a peer also initiates connections to other peers. Their locations are obtained from the tracker.

- **Internal status**. A peer is a large piece of software. We show the set of information as follows.

  - *The torrent.* This specifies the file that a peer is downloading. Remember, we limit the number of torrents that a peer can download to 1.

  - *List of peers.* The list of peers is obtained from the tracker. Note that the list of peers must be updated periodically because peers can join as well as leave.

– *Downloading file bitmap.* For each downloading file, the peer should construct a bitmap which memorizes the chunks that have already been downloaded.

A chunk is a part of a downloading file. Except the last chunk of a file, **every chunk must be of size 256KB**. Whenever the peer has downloaded a chunk, the corresponding bitmap has to be updated.

It is important to note that a user may terminate the peer program before a file has been downloaded completely. For the sake of convenience, **all downloads are then considered finished**. Hence, you are not required to save the chunk bitmap to any permanent storage.

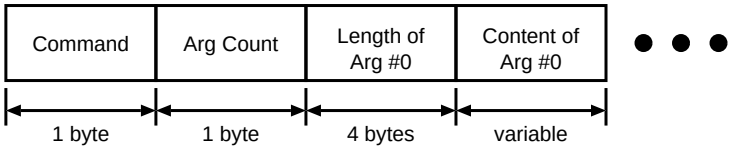
– *Other peers' file bitmaps.* For each peer in the peer list (and we call such a peer the *neighboring peer*), the downloading file bitmap from the neighboring peer should be obtained. Then, by using other peers' bitmap, the peer can determine which chunks from which peers it should download. The set of other peers' bitmaps must be updated periodically. On the other hand, it is not necessary to store the bitmaps into the permanent storage.

- **UI**. You should provide an interface for the user to invoke the following commands:

Command	Description
`add`	Add a new torrent. The user should then supply a torrent file to the peer program. Then, the new download task should begin.
`seed`	Become the source of a newly shared file. The user should supply a torrent file as well as the shared file. Then, the peer will begin uploading chunks to peers.
`stop`	Stop downloading and uploading of an existing torrent. This command can stop both a finished and an unfinished download tasks. All the connections concerning that particular task should be closed, and the peer should refuse further uploading requests. Last but not least, the peer should unregister itself from the corresponding tracker.
`resume`	Resuming the download and the upload of an existing torrent. If the torrent is an unfinished download task, then the download resumes. If the peer has become a seed, then the peer should resume the upload service. By the way, since the peer has to register at the tracker again, the listening port is not required to be unchanged.
`progress`	Print the progress of the downloads in term of percentages and show whether a download is in progress, stopped, or completed.
`peer`	Print the IP addresses and the listening port numbers of the peers for each downloading torrent (not for a stopped torrent).
`exit`	Unregister from the corresponding trackers, terminate all connections, and terminate the peer process.

1.2 Protocols

As a generic requirement, all the application-layer protocols are implemented with TCP. You have to follow our design of the application-layer protocols. All protocols follow a generic design, shown in the following:



1.2.1 Peer-to-Tracker protocol

The peer-to-tracker protocol is a set of request-response messages. The protocol is **non-persistent**, meaning that the replying host closes the connection after a response has been sent.

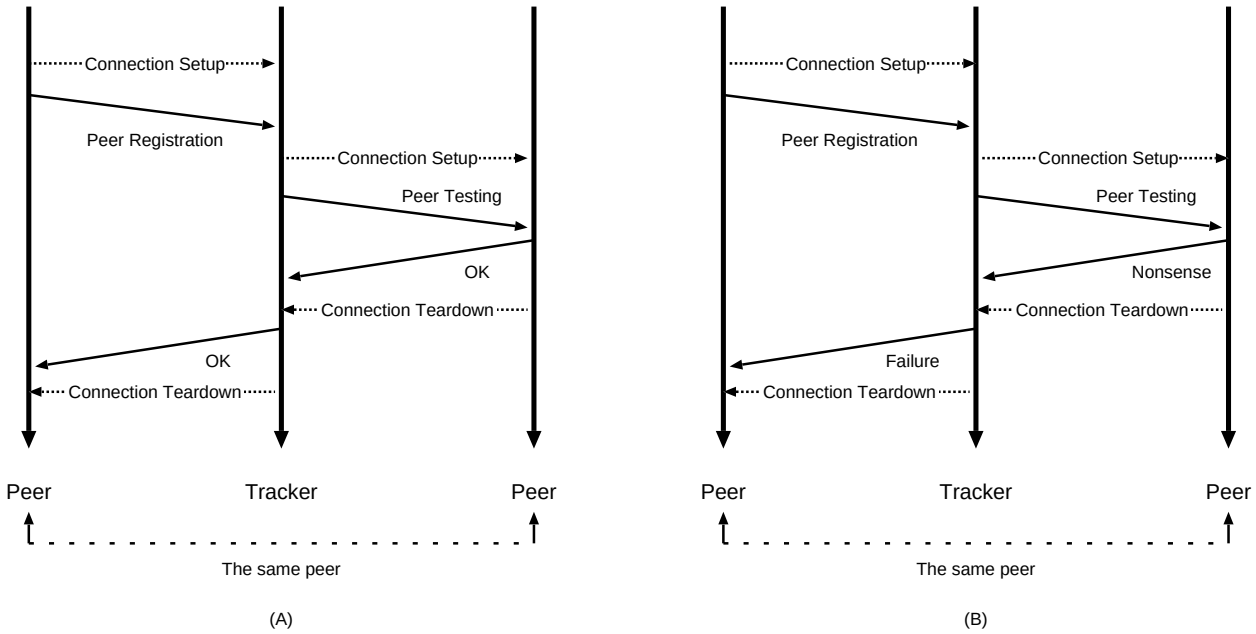


Figure 2: (A) and (B) shows a successful and a failure peer registration scenarios, respectively. For (B), it is not important which side closes the TCP connection.

- **Peer registration**. This function allows a peer to tell the tracker about its presence. The registration is illustrated in Figure 2.
  - **Src & Dest**: from peer to tracker.
  - **Command**: 0x01, the peer-registration request.
  - **Argument count**: 3.
  - **First argument**:
    - Length: 4 bytes.
    - Content: IP address of the requesting peer.
  - **Second argument**:
    - Length: 2 bytes.
    - Content: Listening port of the requesting peer.
  - **Third argument**:
    - Length: 4 bytes.
    - Content: Torrent ID.

Note that the same peer can register multiple times. Hence, the tracker will overwrite its internal data. If the tracker confirms the presence of the peer (with the peer testing function below), it replies with the following message:

  - **Src & Dest**: from tracker to peer.
  - **Command**: 0x11, the peer-registration OK response.
  - **Argument count**: 0.

Else, the tracker replies with the following message

  - **Src & Dest**: from tracker to peer.
  - **Command**: 0x21, the peer-registration failure response.
  - **Argument count**: 0.

- **Peer testing**. This function allows a tracker to test if the location specified by a peer is correct or not.

  - **Src & Dest**: from tracker to peer.
  - **Command**: 0x02, the peer-testing request.
  - **Argument count**: 0.

  If the remote host is a peer, then the peer should give the following reply.

  - **Src & Dest**: from peer to tracker.
  - **Command**: 0x12, the peer-testing OK response
  - **Argument count**: 0.

  Replies other than the peer-testing OK response (including connection failure) will trigger the tracker replies the registering peer with the *peer-registration failure response*.

- **Peer unregistration**. This function allows a peer to remove itself from the tracker and is triggered when a peer stops working on a torrent or it terminates. The input is same as the case of the registration.

  - **Src & Dest**: from peer to tracker.
  - **Command**: 0x03, the peer-unregistration request.
  - **Argument count**: 3.

  - **First argument**:
    * Length: 4 bytes.
    * Content: IP address of the requesting peer.
  - **Second argument**:
    * Length: 2 bytes.
    * Content: Listening port of the requesting peer.
  - **Third argument**:
    * Length: 4 bytes.
    * Content: Torrent ID.

  If an entry corresponding to the unregistration message is found, the record will be removed and the tracker replies with a peer-unregistration OK response.

  - **Src & Dest**: from tracker to peer.
  - **Command**: 0x13, the peer-unregistration OK response.
  - **Argument count**: 0.

  Else, a failure message should be sent.

  - **Src & Dest**: from tracker to peer.
  - **Command**: 0x23, the peer-unregistration failure response.
  - **Argument count**: 0.

- **Peer list download**. This function allows a peer to download the list of the peers which are downloading a particular torrent. A peer should present **the torrent ID** in order to download the list. Note that this action should be triggered by each peer periodically. For the sake of a quick demonstration, the period should be 30 seconds.

  - **Src & Dest**: from peer to tracker.
  - **Command**: 0x04, the peer-list request.
  - **Argument count**: 1.
  - **First argument**:
    * Length: 4 bytes.
    * Content: Torrent ID.

  This request always gives a successful response.

- **Src & Dest**: from tracker to peer.
- **Command**: 0x14, the peer-list OK response.
- **Argument count**: n, where n means the number of entries in the peer list, and $n \geq 0$.
- **For each argument**:
  - Length: 6 bytes.
  - Content: The first 4 bytes carry the IP address and the last 2 bytes carry the listening port number.

1.2.2 Peer-to-Peer protocol

There are two kinds of data sending between peers: the bitmaps and the date chunks.

- **Bitmap retrieval protocol**. This is a non-persistent, request-response protocol. In the following, we define the two roles: the *requesting peer* and the *responding peer*.
  - *Requesting peer*. It starts the TCP connection and presents the torrent ID in the request message.
    - **Command**: 0x05, the bitmap-retrieval request.
    - **Argument count**: 1.
    - **First argument**:
      - Length: 4 bytes.
      - Content: Torrent ID.
  - *Responding peer*. If the requesting torrent ID is the one that the peer is downloading (or uploading), it responds with the downloading file bitmap of the target torrent.
    - **Command**: 0x15, the bitmap-retrieval OK response.
    - **Argument count**: 1.
    - **First argument**:
      - Length: variable.
      - Content: a series of bits.

    Else, the responding peer should return a failure message.
    - **Command**: 0x25, the bitmap-retrieval failure response.
    - **Argument count**: 0.

  Since the file size is already known to both peers, although the length in the first argument the bitmap-retrieval OK response is variable, such a length should always be :

$$\left\lceil \frac{\text{File length in terms of bytes}}{256 \times 1024} \right\rceil \ / \ 8 \ \text{ bytes.}$$

Note that the retrieval should be triggered periodically. Again, for the sake of a quick demonstration, the period should be 30 seconds. You may find the following approach effective:

1. Retrieve the peer list from the tracker.
2. For each peer in the peer list, retrieve the bitmap.
3. Wait for 30 seconds and then go back to the first step.

- **Chunk retrieval**. Again, this is a request-response protocol. However, you are recommended to implement it as a persistent protocol for the sake of (TCP) performance.

  Requesting peer. It presents the torrent ID and the file offset in the request message. Upon the arrival of the response message, the file bitmap of the requesting peer should be updated accordingly.

  - **Command**: 0x06, the chunk-retrieval request.
  - **Argument count**: 2.
  - **First argument**:
    - Length: 4 bytes.
    - Content: Torrent ID.
  - **Second argument**:
    - Length: 4 bytes.
    - Content: File offset in terms of bytes.

  Responding peer. Later in the specification, you will find that you have to implement three versions of peers: the normal peers, the download-only peers, and the sub-seed peers.

  - For each normal peer and sub-seed peer, it responds with the 256KB chunk.
    - **Command**: 0x16, the chunk-retrieval OK request.
    - **Argument count**: 1.
    - **First argument**:
      - Length: usually 256×1024 bytes, except for the last chunk.
      - Content: File chunk content.

- For each download-only peer, it will not respond with any chunk.
    - **Command**: 0x26, the chunk-retrieval failure request.
    - **Argument count**: 0.

1.3 Peer's strategy

To reduce the workload, not all the materials covered in the lectures are required to be implemented.

1.3.1 Peer downloading strategy

The only strategy you are required to implement is the **rarest-first strategy**. Under our assignment setting (with only 10 peers), this is feasible to implement the rarest-first strategy alone.

On the other hand, you should maintain **as few connections as possible** between any two peers. There is no sense for a peer to open many connections to download many chunks at the same time because the overhead, e.g., the opening and the closing of the threads and the sockets, would outweigh the gain from pipelining the download.

1.3.2 Peer uploading strategy

We would implement none of the uploading strategy taught in the lectures. Except the download-only peer, every peer should answer every download request.

2 Deliverables

You have to implement the deliverables in either the C or the C++ programming languages. The UIs are text-based and the programs should be running on Linux only. For each program that you have to deliver, you are free to choose the way to fill in inputs: either using program arguments or using standard inputs. On the other hand, the UI should be as responsive as possible. Every command is expected to be carried out immediately.

2.1 Download-only peer (30%)

A download-only peer implements all the stated features of a peer except the upload function and the "`seed`" command. That means this peer should never upload any chunks to any peers.

2.2 Normal peer (50%)

A normal peer implements all the features stated previously. In addition, you have to implement two modes of a normal peer: the seed mode and the downloader mode.

- *Seed mode*. It means that either the peer has finished downloading a file or it is the source of a shared file.
  - If the peer is the source of a shared file, then the user should use the "`seed`" command to start the upload task.
  - If the peer is a normal downloader but it has finished downloading the torrent, then it also becomes a source of the shared file.

  For both conditions, the peer would not download any bitmaps nor chunks from other peers. Instead, it just uploads bitmaps and chunks to other peers.
- *Downloader mode*. It means the peer has unfinished download tasks. It will exchange downloading file bitmaps and chunks with other peers.

2.3 Sub-seed peer (20%)

The sub-seed peer program is a upload-only peer. However, it uploads a subset of chunks only. This peer is essential for us (and yourself) to test the rarest-first strategy and the ability of the normal peer to complete a download without a seed. We list the features of a sub-seed peer as follows.

- It does not download any bitmaps nor chunks from other peers.

- It uploads its bitmaps and its chunks to other peers.
- It does not implement "`add`", "`seed`", "`peer`", nor "`progress`" commands.
- It provides a new command called "`subseed`". The "`subseed`" command is similar to the "`seed`" command except that the peer is not uploading a complete set of chunks.

We expect the following interactive input and output for the "`subseed`" command.

```
[Subseed Peer] >> subseed 001.torrent
    There are 100 chunks in 001.torrent
    Which chunks to upload? (start counting from 0)
    [type . to stop] >> 0-50
    [type . to stop] >> 52-99
    [type . to stop] >> .
    Confirm (Y/N) >> Y
    Start uploading 99 chunks.
[Subseed Peer] >> _
```

Submission

Please refer to the submission guidelines on our course homepage:

`http://www.cse.cuhk.edu.hk/~csci4430s/`

Deadline: 23:59, March 18, 2012 (Sunday).