CS 357: Algorithmic Game Theory **Spring 2022**

BitTorrent Simulator

Instructor: Shikha Singh

This document describes the provided BitTorrent simulator[1].

General Structure

- Time proceeds in integer rounds, starting at 0.
- There is a single file being shared, composed to `numPieces` pieces, each of which is composed of `blocksPerPiece` blocks.
- The simulation can be run in a population of peers which should include at least one `Seed`. Seeds start with all the pieces of the file and have maximum upload bandwidth. Other peers start with none of the pieces.
- Each peer has an upload bandwidth, measured in blocks per round. This is chosen uniformly at random from a user-specified range (except for seeds, who have the maximum bandwidth). Download bandwidth is unlimited.
- Each round proceeds as follows:
  - Each peer must provide a list of `request` objects, each asking for blocks of a particular piece from a particular peer. This is returned by the `requests` function.
  - Each peer is passed the requests it has received in the current round, and asked to return a list of `upload` objects which specify who they wish to upload to, and how much of its upload bandwidth to allocate to any given peer. This is returned by the `uploads` function.
  - The simulator checks these lists for consistency, that is, requests must ask for pieces the peer has, uploads must add up to no more than the peer's bandwidth cap, etc.
  - Peers who are being uploaded to get their blocks. Multiple requests are satisfied in order until the shared bandwidth is exhausted. For example, if peer 1 requests the last 7 blocks of piece 12, and also piece 11 starting at block 0, and peer 2 includes `Upload(fromId=2, toId=1, bw=10)` in its list of uploads, peer 1 will get the last 7 blocks of piece 12, and the first 3 of piece 11.
  - The events of each round (uploads and downloads performed) are saved in the `history` object, which is available in future rounds.
- The simulation ends once all peers have the pieces or the maximum number of rounds `maxRounds` is exceeded.

[1]Acknowledgement: the simulator has been adapted from the courses David Parkes and Sven Seuken.

Starter Code and Testing

Python3. Make sure you have Python 3.7 or above installed on your machine.

Starter code. You have been given the following python files as your stater code: `sim.py`, `history.py`, `messages.py`, `peer.py`, `seed.py`, `stats.py`, `util.py`, `rand.py`, `freerider.py` and `bittorrent.py`. Among these, you will only be editing `bittorrent.py` but it is a good idea to familiarize yourself with the code in `history.py` and `messages.py`.

Testing. The following command lists command-line arguments that will help in testing.

```
 python3 sim.py -h
```

An initial test command to run is the following. It sets the simulation parameters and runs the simulation on three peers: 1 seed and two random peers (code in `rand.py`).

```
python3 sim.py --loglevel=debug --numPieces=2 --blocksPerPiece=2
--minBw=1 --maxBw=3 --maxRound=5 Seed,1 Rand,2
```

Make sure you understand the output and how the simulation is proceeding. You can set `loglevel` to `debug` when debugging the code, and to `info` if you want a less verbose output. The parameter `--iters` lets you run the simulation multiple times and extract summary statistics from all the runs.

A good setting of parameter for testing the BitTorrent client is the following, but you can try varying parameters.

```
python3 sim.py --loglevel=debug --numPieces=128 --blocksPerPiece=32 --minBw=16
--maxBw=64 --maxRound=1000 --iters=32 Seed,2 BitTorrent,8
```

Free Rider and Random

You are given two simple clients to start with, Rand and Freerider, implemented in `rand.py` and `freerider.py`. They both share the same torrent requesting protocol but differ in how they offer uploads: Rand (short for random), as the name suggests unchokes random requesting clients and uploads to them, while Freerider does not upload to anyone and just leeches off them.

Compare how the freerider client performs when in a population of random clients. A suggested test to compare the two clients:

```
python3 sim.py --loglevel=debug --numPieces=128 --blocksPerPiece=16 --minBw=16
--maxBw=32 --maxRound=1000 --iters=10 Seed,1 Rand,4 Freerider,1
```

How does the average download time and upload bandwidth of the freerider client (round the client completes download) compare to the random clients?

BitTorrent Reference Client

You can implement the reference BitTorrent client in `bittorrent.py` by completing its `requests` and `uploads` function.

In the starter, the file is populated with the code from `rand.py`, just to illustrate how the simple client works. You need to change the code to implement the reference client.

(a) **Requests protocol:** Implement the rarest-first request strategy, which requests pieces with the lowest availability. Thus, you must associate each piece you need with its *rarity* (number of peers who have it) and sort and request based on its rarity.[2]

Note that the client is only allowed to make up to `maxRequests` requests to each peer. If you make any assumptions in your implementation, mention them in the write up.

(b) **Upload protocol:** Implement the Tit-for-Tat (reciprocation) along with optimistic unchoking strategy:

- Assuming 4 unchoke slots, the reference client makes a new decision every round: unchoking 3 peers from whom it downloaded the most data in the past round (can be accessed via `history`).
- Every 3 rounds, the reference client allocates an additional, optimistic unchoking slot, to a random peer (and leaves this peer unchoked for 3 rounds).

If you make any assumptions in your implementation, mention them in the write up.

As a warm up, compare the performance of the BitTorrent client against Freerider in terms of the average download time (completion time).

It is good practice to document the test parameters you use in any comparison and provide explanations to interpret the trends you observe. Some helpful starting question to explore in your empirical analysis are given below.

(a) How do the BitTorrent clients do in a symmetric population of other BitTorrent clients? How does this compare to the performance of the freerider client in the population of other freerider clients?

(b) How does a single freerider client perform in the population of BitTorrent clients? Sample test (may take a while to finish because of the swarm size):

```
python3 sim.py --loglevel=info --numPieces=128 --blocksPerPiece=16 --minBw=16
--maxBw=32 --maxRound=1000 --iters=32 Seed,2 Freerider,1 BitTorrent,9
```

Does the swarm size matter? Compare the performance of freerider in a small neighborhood of only 4 BitTorrent clients vs say 9 BitTorrent clients. What do you notice?

[2]If many pieces have the same rarity (say lots of pieces with only 1 peer who has it) you may break symmetry by randomizing among them.