CS 136 Assignment 2: Peer-to-Peer BitTorrent Simulation

Andy Zhuo and Ben Ray

Credits: office hours with Dylan, Max, Jennifer, Luca, and Alan.
Submission: one late day used.

Problem Set

1. Designing your clients:

   (a) *BitTorrent.*

   Submitted on Gradescope.

   (b) *BitTyrant.*

   Submitted on Gradescope.

   (c) *PropShare.*

   Submitted on Gradescope.

   (d) *Tourney.*

   Submitted on Gradescope.

2. Analysis: Provide your answers based on test runs with the clients you programmed. When asked for comparative performance results, provide some evidence (e.g., simple statistics) for the results you are reporting/describing.

   (a) For the standard client explain what assumptions or decisions you had to make beyond those specified in Chapter 5. (For example, details about unblocking that are not specified in Chapter 5, and how to handle possible corner cases.)

Our standard client, like all clients was created from a `requests` method and an `uploads` method.

Our `requests` method made few assumptions beyond the specification in chapter 5. The standard client employs a rarest-first algorithm to decide which pieces to request among those held by a peer and those that it does not have. We made a decision to break ties between pieces that had the same rarity randomly, although this is a fairly standard procedure.

Our `uploads` method made a few necessary assumptions beyond the specification:

i. The default behavior of the standard client is to consider the average download rate from peers in the previous 20 seconds (two rounds in our case) before unblocking. However, in rounds 0 and 1, two full rounds have not yet been completed. We, therefore, assume that no clients are unblocked in round 0 (an obvious choice, given that our client will not yet have any pieces to give away), and in round 1, we unblock peers based only on the history from the previous round, breaking ties randomly (this means that our client favors those who altruistically unblocked us, without having any pieces themselves, in round 0!)

ii. A further default behavior of the standard client is to split bandwidth into four equal parts, assigning three parts to regularly unblocked peers and one part to an optimistically unblocked peer. In order to deal with the edge case that there are less than four interested peers, we assume that our bandwidth gets equally split between all peers, instead of giving each peer one quarter of our bandwidth and having some left over. Thus, our bandwidth is always fully assigned to unblocked peers every round. There is one edge case / exception to this: if there are no interested peers and no peer has been assigned by optimistic unblocking (in rounds 0 and 1), no peer is unblocked, and no bandwidth is allocated.

iii. If there are less than four interested peers, our standard client will not assign a peer for optimistic unblocking. Alternatively, if our client optimistically unblocked a client three rounds ago, and in the current round there are less than four interested peers, the optimistically unblocked client will stay the same as before for a further three rounds (or until there are at least four interested peers!)

(b) Write a concise summary of the strategies you used for the tournament client, and why you chose them.

We identified that our standard client outperformed the tyrant and propshare clients in a population of mixed clients. We, therefore, chose the standard client (with the assumptions described above) as our baseline design.

We made three key improvements to this baseline design:

i. **Rarest-first optimistic unblocking**: Instead of randomly unblocking an interested peer, chosen from those who had not already been regularly unblocked, we optimistically unblocked the peer from this set, who possessed the rarest piece, breaking ties randomly. This was calculated using the same algorithm as in the `requests` method. We also chose to optimistically unblock a new peer in this way every round instead of every third round. Thus, we are no longer deliberately seeding those who have not received their first pieces, like the standard client. Instead, we are seeking to build cooperation from other clients who have rare pieces that we want.

ii. **Punishment**: if a peer does not unblock us following requests from us in two consecutive rounds, we remove them from our pool of possible peers to upload to, until they next unblock a request from us. Thus, we increase the probability that the peers we unblock are likely to reciprocate when we request pieces from them. It is important to note that our strategy is more like the *Punisher* than *Grim Trigger* in Prisoner's Dilemma. This is intentional, so that we do not isolate ourselves from the swarm, by expelling someone forever from our pool of possible peers to upload to. In other words, we will forgive two unmet requests in a row if the peer unblocks one of our requests in future.

iii. **No seeder state**: this minor adjustment meant that we do not help other players once we have collected all our pieces. Although this only affects the clients who perform worse than us in the tournament, we decided that this selfish policy would likely be adopted by other members of our class!

(c) Outperforming the standard client:

Look at the time it takes to get the file out to all clients (i.e., when does the last client complete downloading the whole file), as well as the average download time for the individual clients.

For the following comparisons, we are using a total population of $n = 10$ clients. As a point of reference, we run the following simulation

```
python3 sim.py --iters=20 --loglevel=info --num-pieces=128
--blocks-per-piece=32 --min-bw=16 --max-bw=64 --max-round=1000
Seed,2 BarazStd, 10
```

which runs 20 iterations with 2 seeds and $n = 10$ standard clients, yielding the following results:

```
======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
BarazStd6: 2414.2 (915.4)
BarazStd9: 2485.2 (1048.2)
BarazStd8: 2557.9 (998.7)
BarazStd4: 2658.1 (1000.8)
BarazStd7: 2682.8 (1072.6)
BarazStd5: 2770.9 (844.1)
BarazStd2: 2951.3 (990.9)
BarazStd1: 2992.2 (924.1)
BarazStd3: 3222.9 (976.5)
BarazStd0: 3605.1 (1110.3)
Seed1: 6220.9 (353.3)
Seed0: 6398.4 (389.2)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
BarazStd3: 97.35 (6.6332495807108)
BarazStd0: 97.5 (6.928203230275509)
BarazStd2: 98.4 (7.0710678118654755)
BarazStd1: 99.95 (7.14142842854285)
BarazStd4: 100.8 (7.0)
BarazStd9: 102.15 (8.774964387392123)
BarazStd5: 102.45 (5.0990195135927845)
BarazStd7: 102.45 (8.660254037844387)
BarazStd6: 102.7 (8.717797887081348)
BarazStd8: 102.85 (7.615773105863909)
```

Continued on next page...

In this population with just standard clients, the clients take an average of 100.66 rounds to acquire all their file pieces. The earliest standard client finishes at an average of 97.35 rounds, and the last standard client finishes at an average of 102.85 rounds.

i. How does the *BitTyrant* client do in a population of standard clients?

Running the same simulation with 9 standard clients and 1 BitTyrant client yields:

```
======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
BarazStd5: 2508.0 (990.7)
BarazStd7: 2549.4 (991.6)
BarazStd8: 2615.5 (808.9)
BarazTyrant0: 2654.9 (817.6)
BarazStd3: 2897.7 (1073.2)
BarazStd2: 2920.6 (1050.4)
BarazStd6: 2956.4 (980.6)
BarazStd1: 2966.9 (1416.2)
BarazStd4: 3096.5 (1168.7)
BarazStd0: 3129.3 (1230.0)
Seed1: 6252.2 (363.3)
Seed0: 6412.6 (384.3)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
BarazStd3: 99.3 (7.615773105863909)
BarazStd4: 99.45 (8.12403840463596)
BarazStd0: 100.15 (7.54983443527075)
BarazStd6: 100.75 (5.291502622129181)
BarazStd2: 100.9 (7.615773105863909)
BarazStd1: 101.45 (9.539392014169456)
BarazStd8: 101.65 (5.656854249492381)
BarazStd5: 102.8 (8.306623862918075)
BarazStd7: 102.95 (9.16515138991168)
BarazTyrant0: 105.35 (6.082762530298219)
```

Continued on next page...

In this population with a single tyrant client against 9 standard clients, the clients take an average of 101.48 rounds to acquire all their file pieces, which is about 0.8% slower than the simulation with exclusively standard clients, so the addition of a BitTyrant client seems to have slowed everyone else down. The earliest standard client now finishes at an average of 99.3 rounds, about 2% longer than the standard simulation. The BitTyrant client finishes last at an average of 105.35 rounds, which is about 2.4% slower than the last standard client in the baseline simulation.

ii. How does the *Tourney* client do in a population of standard clients?

```
======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
BarazStd3: 2533.7 (1012.1)
BarazStd8: 2673.2 (1045.3)
BarazStd7: 2675.3 (843.2)
BarazStd0: 2699.7 (1120.0)
BarazStd6: 2718.8 (962.7)
BarazStd5: 2768.3 (1050.2)
BarazTourney0: 2829.6 (1061.3)
BarazStd4: 2933.2 (938.1)
BarazStd1: 3263.7 (990.5)
BarazStd2: 3404.3 (1064.2)
Seed1: 6144.8 (298.5)
Seed0: 6315.4 (330.7)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
BarazStd1: 97.4 (6.244997998398398)
BarazStd2: 97.5 (4.898979485566356)
BarazTourney0: 99.8 (8.18535277187245)
BarazStd6: 99.8 (7.681145747868608)
BarazStd8: 100.05 (7.745966692414834)
BarazStd4: 100.1 (6.855654600401044)
BarazStd3: 100.45 (7.874007874011811)
BarazStd7: 100.6 (6.782329983125268)
BarazStd5: 101.95 (4.47213595499958)
BarazStd0: 102.4 (7.14142842854285)
```

In this population with a single Tourney client against 9 standard clients, the clients take an average of 100.01 rounds to acquire all their file pieces, which is about 0.7% faster than the simulation with exclusively standard clients, so the addition of a Tourney client seems to have lubricated the distributions. The earliest standard client now finishes at an average of 97.4 rounds, which is about on-par with the 97.35 rounds that the earliest standard client took to finish in the standard-only simulation. The last standard client to finish here took an average of 102.4 rounds, which is about 0.4% faster than the last standard client in the baseline simulation. Unlike BitTyrant and PropShare, which always performed worst in their respective simulations against standard, our implementation of tourney finished somewhere in the middle, and in the above case, 3rd.

iii. How does the *PropShare* client do in a population of standard clients?

Running the simulation with 9 standard clients and 1 PropShare client yields:

```
======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
BarazStd7: 2638.2 (840.4)
BarazStd4: 2714.8 (986.0)
BarazStd6: 2730.5 (950.7)
BarazPropShare0: 2746.3 (887.7)
BarazStd2: 2769.4 (962.2)
BarazStd8: 2779.1 (992.3)
BarazStd5: 2998.8 (940.8)
BarazStd1: 3001.3 (963.4)
BarazStd3: 3035.3 (800.0)
BarazStd0: 3244.3 (1152.5)
Seed1: 6045.2 (271.3)
Seed0: 6256.9 (291.1)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
BarazStd0: 96.3 (6.782329983125268)
BarazStd8: 97.5 (7.937253933193772)
BarazStd5: 98.85 (5.744562646538029)
BarazStd1: 99.2 (7.211102550927978)
BarazStd4: 99.2 (7.54983443527075)
BarazStd3: 99.45 (6.244997998398398)
BarazStd2: 99.55 (6.48074069840786)
BarazStd6: 99.7 (5.656854249492381)
BarazStd7: 99.8 (5.830951894845301)
BarazPropShare0: 100.2 (8.306623862918075)
```

In this population with a single PropShare client against 9 standard clients, all the clients take an average of 98.98 rounds to acquire all their file pieces, which is about 1.7% faster than the simulation with exclusively standard clients, so the addition of a PropShare client seems to have lubricated the sharing among clients and reduced the number of rounds to completion. The earliest client, which is still a standard client, now finishes at an average of 96.3 rounds, about 1.2% faster than the standard simulation. The PropShare client still finishes last, but at an average of 100.2 rounds, which is about 2.6% faster than the last standard client in the baseline simulation and 4.9% faster than the BitTyrant client that finished last when it was the sole non-standard client.

(d) Overall performance of populations:

i. How does a population of only *BitTyrant* clients perform?

Running the simulation with the same parameters as above on 10 BitTyrant clients yields:

```
======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
BarazTyrant8: 2415.4 (814.5)
BarazTyrant9: 2491.0 (942.2)
BarazTyrant4: 2565.6 (934.0)
BarazTyrant2: 2696.9 (717.1)
BarazTyrant6: 2711.4 (898.1)
BarazTyrant7: 2867.6 (825.4)
BarazTyrant3: 3063.1 (921.8)
BarazTyrant5: 3190.0 (1004.0)
BarazTyrant0: 3217.9 (1029.7)
BarazTyrant1: 3395.2 (1289.2)
Seed1: 6069.8 (329.9)
Seed0: 6276.1 (375.4)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
BarazTyrant0: 95.5 (7.810249675906654)
BarazTyrant1: 97.5 (9.055385138137417)
BarazTyrant5: 98.85 (9.746794344808963)
BarazTyrant3: 99.1 (10.14889156509222)
BarazTyrant4: 100.1 (7.416198487095663)
BarazTyrant9: 100.1 (10.14889156509222)
BarazTyrant2: 100.55 (8.54400374531753)
BarazTyrant6: 100.65 (9.0)
BarazTyrant8: 101.25 (8.306623862918075)
BarazTyrant7: 102.2 (7.280109889280518)
```

In this population with 10 BitTyrant clients, all the clients take an average of 99.58 rounds to acquire all their file pieces, which is about 1.1% faster than the simulation with exclusively standard clients. Thus, while a single BitTyrant performs poorly among other standard clients, a population with just BitTyrant clients appears to outperform the standard counterpart. The earliest BitTyrant client finishes at an average of 95.5 rounds, about 1.9% faster than the fastest standard client in the standard simulation. The last BitTyrant client finishes at an average of 102.2 rounds, just about 0.6% faster than the last standard client.

ii. What about a population of only *Tourney* clients?

```
======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
BarazTourney9: 2454.7 (833.0)
BarazTourney7: 2564.4 (933.5)
BarazTourney0: 2719.6 (939.8)
BarazTourney6: 2755.4 (857.3)
BarazTourney1: 2823.8 (905.3)
BarazTourney5: 2909.8 (1161.8)
BarazTourney4: 2965.1 (1289.9)
BarazTourney8: 2976.3 (852.4)
BarazTourney2: 3023.6 (1484.0)
BarazTourney3: 3030.7 (944.6)
Seed1: 6255.0 (327.5)
Seed0: 6481.6 (367.5)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
BarazTourney4: 99.9 (6.782329983125268)
BarazTourney5: 99.9 (9.797958971132712)
BarazTourney3: 100.05 (8.426149773176359)
BarazTourney8: 100.6 (8.0)
BarazTourney2: 101.2 (7.14142842854285)
BarazTourney1: 101.65 (7.211102550927978)
BarazTourney6: 102.4 (6.557438524302)
BarazTourney7: 102.45 (9.433981132056603)
BarazTourney0: 102.5 (8.246211251235321)
BarazTourney9: 103.45 (8.0)
```

In this population with 10 Tourney clients, all the clients take an average of 101.41 rounds to acquire all their file pieces, which is about 0.7% slower than the simulation with exclusively standard clients. The earliest Tourney client finishes at an average of 99.9 rounds, about 2.6% slower than the fastest standard client in the standard simulation. The last Tourney client finishes at an average of 102.2 rounds, just about 0.6% slower than the last standard client. Overall, the Tourney-only simulation finishes a bit more slowly than both the standard-only simulations and the BitTyrant-only simulations.

iii. How does a population of only *PropShare* clients do?

```
======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
BarazPropShare7: 2401.3 (936.2)
BarazPropShare5: 2456.2 (1002.7)
BarazPropShare4: 2609.1 (1009.8)
BarazPropShare3: 2662.4 (935.1)
BarazPropShare0: 2766.8 (1118.0)
BarazPropShare9: 2803.4 (875.9)
BarazPropShare2: 2890.5 (1026.6)
BarazPropShare8: 3001.8 (867.2)
BarazPropShare1: 3193.7 (944.8)
BarazPropShare6: 3258.8 (729.4)
Seed1: 6362.2 (424.3)
Seed0: 6553.8 (446.7)
Completion rounds: avg (stddev)
Seed0: 0.0 (0.0)
Seed1: 0.0 (0.0)
BarazPropShare6: 100.2 (9.1104335791443)
BarazPropShare1: 100.65 (10.04987562112089)
BarazPropShare8: 101.75 (10.723805294763608)
BarazPropShare2: 102.45 (7.0)
BarazPropShare9: 102.6 (8.831760866327848)
BarazPropShare0: 103.15 (12.328828005937952)
BarazPropShare3: 103.8 (8.831760866327848)
BarazPropShare4: 104.4 (10.099504938362077)
BarazPropShare5: 104.7 (9.643650760992955)
BarazPropShare7: 106.4 (9.1104335791443)
```

In this population with 10 PropShare clients, the clients take an average of 103.01 rounds to acquire all their file pieces, which is about 2.3% slower than the simulation with exclusively standard clients. Though having a PropShare client lubricated a population with primarily standard clients, a population with just PropShare could not replicate the effect on each other. The earliest PropShare client finishes at an average of 100.2 rounds, about 2.9% slower than the fastest standard client in the standard simulation. The last PropShare client finishes at an average of 106.4 rounds, about 3.5% slower than the last standard client.

(e) Write a paragraph about what you learned from these exercises about BitTorrent, game theory, and programming strategic clients? (We aren't looking for any particular answers here, but are looking for evidence of real reflection.)

The most valuable takeaway from our foray with BitTorrent clients was how different clients affect each other differently in different settings. In particular, we were curious about the fact that the presence of certain strategies in a swarm could help some other strategies while harming others. For example, Tyrant was seen to slow down standard clients, while PropShare clients sped them up, even though they both performed worse than standard in a population of just standard clients. This goes to show that benevolent planners of such BitTorrent networks have an outsized role to play. Would they rather make it easy for one strategy to be much faster than others, or would they rather slow down certain strategies to make them more similar in speed to other strategies? The former could be considered more efficient, while the latter could be considered more equitable.

Finally, we have also noted that strategies in one repeated game are not necessarily generalizable to a similar variant. One of the first drafts of our Tourney client used some of the learnings from the first CS136 tournament, employing a variation of the *Grim Trigger* strategy that worked well in the iterated Prisoner's Dilemma. However, we quickly realized that this would isolate us from many peers in the swarm, and so we switched to a *Punisher*-esque strategy. We are particularly excited to further explore repeated games with more than two players to see whether our learnings from this CS136 tournament are generalizable to other derivations of repeated games.

3. Theory:

(a) State three ways in which the peer-to-peer file sharing game of the BitTorrent network is different from a repeated Prisoner's dilemma.

i. In a BitTorrent network agents must interact with a large number of players in a swarm, meaning they do not necessarily interact with the same peer in every round. In a repeated Prisoner's Dilemma, an agent interacts with the same opponent every round.

ii. The BitTorrent network operates more like a repeated auction than a Prisoner's Dilemma, since peers are competing for each other's upload capacity. As such, "winners of the auction" are the "highest bidders" in each round, rather than those who play the best responses to other players as in the Prisoner's Dilemma. Moreover, different clients in a BitTorrent network may assign different utilities to pieces available in different "auctions" - e.g. they may preference the auctions of "rarer" pieces. By comparison, the Prisoner's Dilemma has the same payoffs in each round (unlike an auction).

iii. BitTorrent network agents can choose what information to reveal to peers, including the holding back information to optimize for a certain goal. For example, some clients wish to extend upload/download relationships with other peers, they may hold back information on rare pieces for a period of time. By contrast, agents in a Prisoner's Dilemma have perfect information about the actions of an opponent. As such, the information asymmetry possible in a BitTorrent network is not seen in a repeated Prisoner's Dilemma.

(b) State three ways in which the BitTorrent reference client is different from the tit-for-tat strategy in a repeated Prisoner's Dilemma.

i. The BitTorrent reference client uses "optimistic unblocking" to unblock another player that has not necessarily unblocked them. You could argue that the *TfT* strategy works like this in the first round of a repeated Prisoner's dilemma (i.e. cooperating with an opponent with no prior evidence that they have cooperated), but thereafter, the *TfT* agent will only cooperate if the opponent cooperated in the previous round.

ii. The BitTorrent reference client chooses its action based on actions taken by others in the last x (in our case, 20) seconds. In this time, multiple rounds (in our case, 2) may have occurred. A *TfT* agent will condition its action solely on the action of its opponent from the last round. You could, therefore, argue that the BitTorrent reference client makes a more informed decision by looking deeper into the history of the game.

iii. The BitTorrent reference client assigns different "utilities" to different pieces, preferring those which are rarer. By comparison, a *TfT* agent has set utilities for its action set.

(c) Explain two reasons why just having a BitTorrent client that is a best response to itself is insufficient for this client to form an equilibrium in a peer-to-peer system. (Hint: you can think about both theoretical reasons and practical reasons. For example, what could happen if there is another client that is also a best-response to itself? What if different users care about different objectives?)

In an equilibrium, there cannot be a useful deviation for any client. Here are two reasons why two clients that are a best response to themselves are insufficient to form an equilibrium in a P2P system:

i. If a client is designed to best respond to another client that is the same, it is unlikely to best respond optimally to another type of client. For example, BitThief was shown to exploit optimistic unblocking by the reference client. Thus, when faced with a BitThief client, the reference client would have an incentive to deviate from its standard procedure of optimistic unblocking.

ii. BitTorrent clients do not necessarily have the same objectives - e.g. the reference client prioritizes rare pieces, while others prioritize the length of their upload/download relationship with a peer. If a client that prioritizes the latter meets a reference client, it can deliberately make the more common pieces available (and hold back the rare pieces). The rare-preferencing client would, thus, benefit from requesting from others who would give them rare pieces sooner, i.e. deviating from its standard procedure.