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

David C. Parkes
School of Engineering and Applied Sciences, Harvard University
Out Fri Sep 22, 2023
Due **5pm** sharp: **Friday Sep 29, 2023**
Submissions to Gradescope

Problem Set

Assignment completed by Jessica Chen and Helen Xiao

1. 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.)

   Solution: For the standard client, we can split our assumptions into two: assumptions made for requests, and assumptions made for uploads. On the requests side, we had the assumption that the reference client would want to maximize the probability of obtaining rare pieces, and assumed the way to do so would be to request from every client with a rare piece for that rare piece, even if this risked requesting a piece from many people at once, which could result in the client obtaining files at a slower rate in terms of rounds yet help the higher guarantee of obtaining rare files. On the uploads side, we assumed that the standard client would set the number of slots to be 4 (self.m = 4), and that the client did not require all 4 slots to be filled: there would be one optimistically unblocked peer, and the rest of the peers for regular unblock were limited to but not required to be the length of one less than the number of slots. We also assumed a random peer who had requested (without being chosen) to be optimistically unblocked.

   We also assumed that the pieces within each client were generally in the same order as for other peers (starting out) so we always shuffled the pieces before requesting to break the cases where pieces are of same rarity to break symmetry. Through our program, we shuffle the needed pieces, the peers, and the intersection of the available and needed pieces.

   (b) Write a concise summary of the strategies you used for the tournament client, and why you chose them.
   Since our reference client showed consistently strong performance in a variety of environments, we decided to adapt the reference client to create our Tourney client. We made three main changes:

i. **Utilize all available upload slots:** In the vanilla standard client, we do not implement additional measures for maximizing bandwidth in the case of too-few reciprocated requests. If there are not enough requests coming from peers that have reciprocated in previous rounds, we simply choose all the reciprocated requests we have and split the bandwidth accordingly. In our modified Tourney client, we give the empty slots as opportunities to non-reciprocating requesters. This behavior is comparatively more altruistic; even though these requesters may not have unblocked us in the past, we still take the initiative to upload to them.

ii. **Decrease change frequency for the optimistic peer:** Instead of changing the optimistic peer every modulo-3 rounds (and whenever the optimistic peer does not request), we increase the number of rounds an optimistic peer may stay for. By decreasing this change frequency, we hope to build more substantive relations with certain randomized peers–we estimate that it may take longer for a mutually altruistic relation to foster. This also help balance our above optimization, which already emphasizes heavily on diversifying the peers we upload to.

iii. **Increase historical consideration for peer download rate:** When sorting peers by decreasing average download rate, we modified to consider more rounds in history. Whereas the vanilla model considered 20 seconds into history, we consider 50 seconds into history. This change does not necessarily work well against an environment of purely tit-for-tat peers, but does help when playing against tyrant-like agents. Generally, if other tournament players are implementing more complex strategies that consider more historical behavior, then this modification helps take such a factor into account as we rank reciprocating peers.

(c) Outperforming the standard client:

```
======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
g41Tyrant0: 460.6  (145.1)
g41Std4: 617.0  (130.4)
g41Std6: 626.8  (165.5)
g41Std7: 673.8  (140.1)
g41Std5: 703.5  (156.3)
g41Std2: 745.8  (137.8)
g41Std8: 746.2  (127.9)
g41Std3: 799.3  (143.2)
g41Std1: 811.8  (169.4)
g41Std0: 854.9  (144.0)
Seed1: 1145.3  (78.0)
Seed0: 2055.0  (94.6)
Completion rounds: avg (stddev)
Seed0: 0.0  (0.0)
Seed1: 0.0  (0.0)
g41Std2: 30.0  (1.7320508075688772)
g41Std0: 30.15  (2.0)
g41Std1: 30.25  (2.23606797749979)
g41Std3: 30.3  (2.23606797749979)
g41Std4: 30.5  (2.23606797749979)
g41Std6: 30.6  (2.0)
g41Std8: 30.85  (2.449489742783178)
g41Std5: 30.9  (1.4142135623730951)
g41Std7: 31.05  (2.0)
g41Tyrant0: 32.75  (1.4142135623730951)
```

Figure 1: BitTyrant in Std Environment

```
======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
g41Std8: 610.1  (164.3)
g41Std2: 652.0  (160.7)
g41Std7: 657.2  (165.7)
g41Std3: 675.5  (146.6)
g41Tourney0: 696.6  (175.0)
g41Std5: 706.8  (177.5)
g41Std6: 732.9  (168.5)
g41Std4: 751.4  (139.3)
g41Std1: 768.0  (156.1)
g41Std0: 810.9  (170.2)
Seed1: 1160.7  (72.8)
Seed0: 2017.9  (115.9)
Completion rounds: avg (stddev)
Seed0: 0.0  (0.0)
Seed1: 0.0  (0.0)
g41Std0: 29.45  (1.7320508075688772)
g41Std1: 29.55  (2.23606797749979)
g41Std2: 29.85  (1.7320508075688772)
g41Std3: 30.65  (2.0)
g41Std6: 30.8  (1.7320508075688772)
g41Std7: 30.85  (2.23606797749979)
g41Tourney0: 30.95  (1.7320508075688772)
g41Std4: 31.0  (1.7320508075688772)
g41Std5: 31.15  (2.23606797749979)
g41Std8: 31.5  (2.449489742783178)
```

Figure 2: Tourney in Std Environment

```
======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
g41Std3: 624.3  (136.5)
g41Std7: 634.3  (140.0)
g41Std5: 660.5  (147.7)
g41Std4: 705.5  (175.5)
g41Std2: 717.0  (152.3)
g41Std8: 720.3  (198.1)
g41Std6: 739.1  (152.3)
g41Std1: 761.4  (202.5)
g41PropShare0: 766.6  (147.2)
g41Std0: 782.4  (186.8)
Seed1: 1143.6  (70.0)
Seed0: 1985.0  (93.8)
Completion rounds: avg (stddev)
Seed0: 0.0  (0.0)
Seed1: 0.0  (0.0)
g41PropShare0: 28.9  (1.7320508075688772)
g41Std1: 29.4  (1.4142135623730951)
g41Std0: 29.7  (2.23606797749979)
g41Std2: 29.7  (2.0)
g41Std4: 29.9  (2.449489742783178)
g41Std5: 30.35  (2.0)
g41Std8: 30.55  (1.7320508075688772)
g41Std3: 30.6  (1.7320508075688772)
g41Std6: 30.65  (2.23606797749979)
g41Std7: 30.7  (1.7320508075688772)
```

Figure 3: PropShare in Std Environment

i. How does the *BitTyrant* client do in a population of standard clients?
Answer: (see the above sreenshots for evidence) We see that the BitTyrant performs worse in a population of standard clients: not only does it upload less in

quantity, but also, the time/number of rounds in which it takes to complete the uploads is smaller. In this sense, we see that BitTyrant does not outperform the population of standard clients (for our simulation) since it is slower in obtaining files. This may be due to our choice of initialization values, which are likely to overestimate the upload and download rates.

ii. How does the *Tourney* client do in a population of standard clients?
Answer: (see the above screenshots for evidence) We see that the Tourney client performs very similarly to the standard client. Generally, the number of rounds taken is about the same as the standard client, and additionally the amount uploaded is the same. While we did increase the number of rounds to consider for historical download rates, as well as enforce always having four separate equally bandwidthed upload slots, the mix of modification goals (increasing diversification of upload peers via optimistic unblocking, but decreasing the frequency of changing the optimistically unblocked peer) may ultimately lead to neutral improvement. In this way, the tourney client does not outperform the standard client since there are so significant or noticeable differences.

iii. How does the *PropShare* client do in a population of standard clients?
Answer: (see the above sreenshots for evidence) We can see that the PropShare client does very well in a population of standard clients: we have that the it takes an average of less rounds than the standard clients, while also uploading about the same if not a better amount than the standard clients. In this sense, PropShare outperorms standard client when there only exist standard clients and PropShare: this may be true because Propshare awards those who upload more to them, and therefore, those clients are more inclined to also upload to them.

Look at the relative ranking of the clients and the percentage improvement (or impairment) in the number of rounds it takes the client to get the complete file.

(d) Overall performance of populations:

```
======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
g41Tyrant3: 606.4  (220.3)
g41Tyrant8: 633.8  (271.5)
g41Tyrant7: 655.5  (150.0)
g41Tyrant5: 687.5  (150.2)
g41Tyrant6: 692.0  (212.3)
g41Tyrant4: 701.3  (208.9)
g41Tyrant9: 710.5  (173.6)
g41Tyrant0: 713.6  (178.5)
g41Tyrant2: 716.6  (120.8)
g41Tyrant1: 746.0  (164.6)
Seed1: 1183.8  (91.3)
Seed0: 2193.0  (153.6)
Completion rounds: avg (stddev)
Seed0: 0.0  (0.0)
Seed1: 0.0  (0.0)
g41Tyrant3: 31.8  (2.23606797749979)
g41Tyrant5: 32.1  (2.8284271247461903)
g41Tyrant9: 32.55  (3.3166247903554)
g41Tyrant1: 32.6  (3.605551275463989)
g41Tyrant7: 32.7  (3.3166247903554)
g41Tyrant8: 33.1  (2.6457513110645907)
g41Tyrant6: 33.25  (3.0)
g41Tyrant2: 33.65  (4.0)
g41Tyrant0: 34.6  (3.4641016151377544)
g41Tyrant4: 34.65  (2.449489742783178)
```

Figure 4: All BitTyrants

```
======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
g41Tourney8: 608.1  (171.5)
g41Tourney7: 616.0  (216.4)
g41Tourney9: 628.5  (173.9)
g41Tourney6: 668.5  (213.8)
g41Tourney2: 693.5  (199.9)
g41Tourney1: 702.3  (218.1)
g41Tourney3: 728.2  (230.0)
g41Tourney5: 731.5  (237.9)
g41Tourney0: 741.4  (233.4)
g41Tourney4: 788.7  (198.7)
Seed1: 1178.9  (100.3)
Seed0: 2154.3  (104.3)
Completion rounds: avg (stddev)
Seed0: 0.0  (0.0)
Seed1: 0.0  (0.0)
g41Tourney1: 30.25  (1.7320508075688772)
g41Tourney0: 30.6  (2.23606797749979)
g41Tourney5: 31.1  (2.6457513110645907)
g41Tourney3: 31.2  (2.449489742783178)
g41Tourney2: 31.4  (2.23606797749979)
g41Tourney4: 31.7  (2.23606797749979)
g41Tourney6: 31.95  (2.23606797749979)
g41Tourney8: 32.1  (2.8284271247461903)
g41Tourney7: 32.4  (2.0)
g41Tourney9: 32.4  (3.0)
```

Figure 5: All Tourneys

```
======== SUMMARY STATS ========
Uploaded blocks: avg (stddev)
g41PropShare9: 646.3  (169.1)
g41PropShare4: 660.8  (145.2)
g41PropShare8: 674.9  (117.6)
g41PropShare6: 675.4  (122.6)
g41PropShare7: 687.0  (194.9)
g41PropShare2: 720.6  (156.3)
g41PropShare3: 742.6  (148.9)
g41PropShare5: 747.9  (129.8)
g41PropShare1: 770.3  (133.0)
g41PropShare0: 852.3  (166.3)
Seed1: 1095.2  (68.0)
Seed0: 1966.7  (110.4)
Completion rounds: avg (stddev)
Seed0: 0.0  (0.0)
Seed1: 0.0  (0.0)
g41PropShare1: 29.3  (2.0)
g41PropShare0: 29.35  (1.7320508075688772)
g41PropShare3: 29.45  (1.7320508075688772)
g41PropShare7: 29.55  (1.7320508075688772)
g41PropShare2: 29.65  (2.0)
g41PropShare6: 29.65  (2.0)
g41PropShare8: 29.65  (1.4142135623730951)
g41PropShare5: 29.7  (2.0)
g41PropShare9: 30.05  (2.0)
g41PropShare4: 30.45  (2.23606797749979)
```

Figure 6: All Propshares

i. How does a population of only *BitTyrant* clients perform? What about a population of only *Tourney* clients?
See above Figures 4, 5, 6 to see full numeric results. A population of only BitTyrant clients moderately well. The final client to complete downloading the whole file completes around Round 34, and the average download time is around Round 32. A population of only Tourney clients performs slightly better, with the last client completing download at Round 32 and the average download time is around Round 31.

ii. How does a population of only *PropShare* clients do?
Our PropShare clients perform best when in a homogeneous environment. The last client to complete download does so at Round 30, and the average download time is around Round 29.5.

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.

(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.)
Personally, we have used torrenting services in the past, especially as high-schoolers who wanted to watch shows without paying for subscriptions. However, we never knew how the protocol worked beyond the basic idea of "peer-to-peer" being a way to decentralize download load. Specifically, we never knew the meaning behind seeding, leeching, and more, and how, for the P2P system to work, there must be some architecture of game theory that incentivizes uploading, encourages convergence, and minimalizes tragedy of the commons.
As someone who only considered being part of the torrenting community as an unaltruistic leecher, reading through the textbook made me check back into the exact client software I had been using. I was afraid that I was still unknowingly using my resources to upload pieces for other peers. Luckily, based on a quick Google search (and now with knowledge of how the protocol works), it seems the qBittorrent client software I used has relatively strong measures to ensure fair usual privacy and bandwidth usage protocols. This is apparently in contrast to uTorrent, or even BitTorrent (the client), which are known to be more susceptible to malware. After this programming assignment, we plan to delve more deeply into how the modern piracy torrenting community is interacting, especially as its popularity declines.

2. 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.
Answer: We see that the peer-to-peer file sharing game of the Bittorrent network is different from a repeated Prisoner's dilemma because:

- While in a repeated Prisoner's dilemma there are only two clients, who will gain a certain utility based only on the other client's play, in the Bittorrent network, instead there is the case that a client is competing against a swarm of other clients,

who are all "outbidding" each other for an upload, and a client's results are affected by multiple other client's actions. Additionally, because of the "multiple blocks of a file" situation that occurs for the BitTorrent network, we have that different clients have different offers to bid and can influence their ability to obtain something based off of how much they bid.

- Additionally, while the prisoners in prisoner's dilemma have similar/the same motivations: to increase utility, in the BitTorrent network, different clients may be optimizing for different things: for example, the BitTyrant is trying to gain as many pieces as possible by increasing connections. On the other hand, other clients may be trying to optimize for number of uploads to others.
- Finally, while in the Prisoner's dilemma, the only available information a prisoner has about another prisoner is the actions (C or D) of the other, in the BitTorrent network, clients can decide what or what not to reveal to other clients, which can effect client's strategies. For example, clients can choose to withhold information about rare-pieces to keep others interested.

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

- We see that the BitTorrent reference client is different from a tit-for-tat strategy since while for the tit-for-tat strategy if one person defects, the other person will also defect, we see that the BitTorrent reference client uses "optimistic unblocking" which will upload to another client even if they had not uploaded to them previously.
- Additionally, because different pieces have different rarities, a BitTorrent reference client holds different "utility gains" for different pieces, and therefore, the payoff matrix for a BitTorrent reference client would be much larger and complex than that of a TFT strategy in repeated Prisoner's dilemma.
- Finally, we see that the BitTorrent reference client does not quite mirror the tit-for-tat strategy since difference clients have different priorities based on what pieces they have or do not have, and therefore, BitTorrent clients can not just take into consideration the information and the actions of other clients from one previous round, but instead require considerations of what pieces they gained from previous rounds and actions of other clients from more than one previous round.

(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?)

For clients in a P2P system to be in equilibrium, no client can have a useful deviation. If all classes of clients are optimized to be best responses to themselves, then it is likely that they can have a useful deviation when playing against a different class of client. Let us consider an environment when all clients are simply best responses to themselves. Then, it is easy for us to choose one client and modify their strategy so that they take advantage of another client's strategy, without risk of the other client's further best response to your change. We can illustrate the two reasons using practical examples.

i. We see that BitThief exploits the reference client's optimistic unblocking by "large-view exploit", which attempts to make as many additional connections as possible, as to increase the chance of being unblocked.

ii. A client can also leverage strategic piece revelation. Generally, sending out the have-pieces messages helps increase the interest that other peers may have in our client. However, knowing that the reference client will prioritize rarest pieces first, our modified client can purposefully give out the most common pieces first and hold onto the rarest pieces, thus keeping trader partners for as long as possible.

The above strategies are useful deviations that a client may implement when in a swarm of other self-best-responding clients. Therefore, we would not necessarily form an equilibrium.