Consistency and Collaboration

Victoria Michalska

Abstract

This paper will be looking at the quantity and quality of exchanges between specific pairings of peers with different or identical protocols in a given P2P environment using heatmaps; holding all of the parameters constant with the exception of the kind of protocols present, different renditions of tit-for-tat's and their strengths will be explored and then exploited in order to potentially create a protocol that will intentionally exploit tit-for-tat in order to reveal its lack of robustness against even simple clients.

1 Introduction

Peer-to-peer (P2P) networks are systems of interacting components with processing and receiving capacities, connected directly to other components without a centralized entity which controls the allocation of resources between peers. These networks exist in the banking industry, the electric power industry, as well as in the world of online file-sharing. This project will be engaging in an empirical study regarding different strategies for specific peers to utilize when engaging with their peers, with some theoretical attempts at explaining the results; in the implementation, there will be a focus on file-sharing systems.

In the world of file-sharing, BitTorrent is the most common protocol. Its approach is primarily based off the idea of tit-for-tat, which is explained in depth in Section 2 of this paper. It has been shown not to be robust; an alternative to this protocol is FairTorrent. It is more generous and forgiving than BitTorrent, meant to utilize the bandwidth that is wasted with BitTorrent when the protocol actively chooses to choke particular peers due to their behavior in prior rounds.

In this project, a provided simulator written up in Python3 will be utilized in order to implement different agents. BitTorrent will be implemented in collaboration with FairTorrent in order to emphasize the respective strengths between the two protocols, and this knowledge is then used in order to create a similar protocol that may similarly exploit BitTorrent and/or FairTorrent in order to yield lower average download times for the given agent.

2 Preliminaries

P2P networks are a common method for recourse distribution; theoretically, these sorts of networks can be expressed as a repeated game, where uploading and downloading have their own respective utilities. This game resembles the prisoner's dilemma, pictured in Figure 1, where there are two options for each player: either the agent can cooperate (C) or dissent (D). In the context of this paper, assume that the benefit of receiving the file is 3 and the cost of uploading is 1. The $\_\_$ symbols are representative of whatever move the given agent had ended up with as a result of the automata's processes regarding the opponent's moves.

		Peer B	
		C	D
Peer A	C	(2,2)	(-1,3)
	D	(3,-1)	(0,0)

Figure 1: Payoff Matrix for P2P Game

This payoff matrix is constant between two peers, but the strategies which a given peer can use to move about this matrix can vary, especially given that upload bandwidth is limited for a given peer– meaning that the number of peers that a given agent can cooperate with in a given round is limited.

One approach is tit-for-tat: a strategy which assumes corporation between two players from the start. In the rounds following, the given player will do what their opponent did in the following round. It can be represented as a push-down automata via the following diagram:

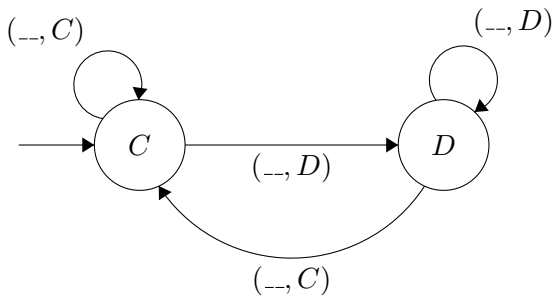

Figure 2: DFA for Tit-for-Tat

3 Related Work

This project was originally inspired by a paper by Khan, Waheed, and Saif, under the title, *BitTorrent for the Less Privileged*. It explored the quantity of exchanges and *unchokes* between different pairs of peers with bandwidths that fell under a uniform distribution. An unchoke, in this context, defined as the choice to upload to a given peer. Khan's paper created heatmaps that visualized the quantity of unchokes between different pairs of peers; given that these peers were of different bandwidths, those with lower bandwidths were often the last to finish downloading the appropriate file. This information highlighted the lack of bandwidth-based clustering from which the lower-bandwidth peers could have benefited by collaborating, and in order to address this, the authors of the paper created BitMate.[2]

While this project will not be focusing on bandwidths, it will be utilizing a similar visualization technique to break tit-for-tat's robustness for strategic players. By looking at the quantity and kind of exchanges in environments with different sets of agents, certain gaps may be addressed and then utilized to take advantage of tit-for-tat.

4 Background on Protocols

Both, BitTorrent and FairTorrent, which this project will be focusing on, have similar request processes: in order to prevent the bottlenecks that occur due to a given piece being the most limited in an environment, the agents will continually request the pieces with the highest rarity, randomly breaking ties.

4.1 BitTorrent

BitTorrent's upload process is its most fundamental component: it will only upload to peers who uploaded to them– in the implementation of it in the simulator, this "past" will be considered to be only the most recent round. The peers that did not upload to them in the most recent round will be choked. When a peer is unchoked, then the given agent is uploading to that peer. A typical BitTorrent has four slots: at most, three of which are occupied by peers that had uploaded to them in the past, and a fourth, which is considered an *optimistic unchoke*. This involves the inclusion of an additional random peer to be unchoked in order to facilitate future uploading of that peer to other peers. The total upload bandwidth will then be split up evenly between the requests selected to be satisfied.[1] This is an efficient method against peers that intend to download without uploading, which is empirically shown in Section 6.2.

4.2 FairTorrent

A major issue with BitTorrent is the wasted bandwidth involved with intentionally choking peers that hadn't uploaded to them in the prior round. In order to create a compromise between tit-for-tat and an efficient use of bandwidth, FairTorrent's algorithm was created. FairTorrent involves the maintenance of a deficit list, where all of the blocks uploaded from the peer to a different given peer are summed and then all of the blocks downloaded from that different given peer to the original peer are subtracted from that sum. This deficit list is then sorted and used by the agent in order to select a peer to whom they will upload: the peer with the lowest deficit will be uploaded to first– if the peer with the lowest deficit has not put in a request, then the peer with the following lowest deficit will be checked; this process will continue until all of the possible bandwidth is filled, or there are no more peers left. When an appropriate request is found, the entire requested piece (or the maximum bytes/blocks possible) will be uploaded to that peer. If there is bandwidth remaining, then another request will be picked up using the same process described earlier.[4]

4.3 Freerider

Freerider is a protocol that explicitly does not upload to any peers. This lack of contribution on the part of this sort of protocol results in a detriment to the overall performance of the network in general by not taking advantage of the equal privileges given to the peers in the P2P network. Its request protocol functions by going through all of the peers and choosing the maximum possible number of random pieces that it still needs from a given peer, in hopes that the given peer will deliver.[3] Tit-for-tat actively addresses this by avoiding uploading to peers that do not reciprocate downloads, making it depend on the `Seed`s for downloads, as seen in Section 6.2.

4.4 Seed

A `Seed` is a node in the network that holds the entirety of the original file, only uploading to its peers depending on its approach. Here, the `Seed` will be choosing random peers to upload to it, functioning exclusively altruistically.

5 Tit-for-tat's Robustness

BitTorrent has been shown not to be particularly robust to strategic peer behavior in the past with approaches more complex than Freerider (described in Section 4.3), due to the implementation of clients like BitTyrant, which utilizes higher quantities of computation (involving calculating which peers have large reciprocation bandwidth, the marginal benefit of an additional peer which will outweigh the cost of reduced reciprocation probability from other peers, as well as calculating the upload contribution needed for a particular peer in order to continue its reciprocation) to exploit the altruism featured in the nuances of BitTorrent's implementation through the inclusion of optimistic unchokes and the even splitting of the bandwidth for BitTorrent's four unchoke slots, as described in Section 4.1 of this paper.[3] However, this is computationally heavy and not particularly accessible to less knowledgeable audiences. Therefore this paper will be attempting to prove tit-for-tat's lack of robustness against simpler clients.

An alternative to BitTorrent when it comes to the implementation of tit-for-tat as a file-sharing protocol is FairTorrent, described in detail in Section 4.2. By not actively maintaining chokes but still preferring to upload to peers who have uploaded to the given agent, FairTorrent does establish a threat to peers who don't upload to it; however, because FairTorrent does acknowledge the full history, rather than just the most recent round, it can be considered more forgiving and should allow for greater collaboration between the peers in the long-term , given the limited upload bandwidth of the agents.

6 Methodology and Contributions

This project will be using a P2P file-exchange simulator run in Python3 in order to collect data. Original implementations of BitTorrent and FairTorrent were written up for the purposes of this experiment, alongside the creation of a function that creates a heatmap in order to visualize the sort of exchanges occurring throughout the duration of the experiment. Given the original P2P simulator, certain adjustments were necessary in order to ensure maximum randomization possible to prevent the presence of patterns developing in the results where there would be none in practice; this is specifically regarding the sorting of the peers alphabetically at certain points in the simulator and this being seen in the results. Afterwards, new functions had to be created in order to facilitate the presentation of data as heatmaps from the simulator, described below in Section 6.1. Other functions were created in addition to these, but these are the most integral ones to understanding how this experiment is run. In the context of this paper, these heatmaps will be displaying primarily the frequency of the number of unchokes, because the major focus of this paper is the quantity of collaboration occurring between peers, as inspired by the Khan paper mentioned in Section 3; however, the number of blocks exchanged may also be featured in the heatmaps when of relevance in the context of this paper.

6.1 Implementation

There are two original functions integral to creating the heatmaps necessary for this paper. They are the following:

- **convertDictList(frequenciesDict)** Featured in `util.py`, this function, given a dictionary where the keys are touples where the first element of the touple is the uploading agent and the second element of the touple is the downloading agent, and where the values are the frequency of the given exchange, will convert it into a list of lists where each of the lists contains the first element of the touple, the second element of the touple, and the corresponding frequency.
- **frequencyMap(frequenciesList, bandwidths)** This takes a list of lists containing the following elements in this exact order: the uploading agent, the downloading agent, and the corresponding frequency. It then uses matplotlib in order to create a heatmap using ishow, representing those frequencies visually using a heatmap where darker squares represent a higher number of exchanges, with a corresponding frequency bar. Note that because a peer cannot upload and download from itself, there will be a white line along the z-axis of the heatmap.

A significant question regarding the approach of this experiment is manner of distribution of bandwidth: should it be uniformly distributed, or equally distributed? Because it is specifically the algorithm type that is being tested, all of the non-`Seed` peers will be given the same bandwidth: that bandwidth being calculated by finding the middle point between the minimum and maximum bandwidth assigned. The `Seed`s will all be assigned the maximum bandwidths.

Regarding BitTorrent, as described in detail in Section 4.1, at round i, chokes were implemented specifically for peers that did not upload to them during around $i - 1$, in contrast to the 6 most recent rounds, in attempt at get as close to the standard definition of tit-for-tat as possible in the context of P2P networks. Furthermore, a maximum of 4 unchoke slots are allowed for any one given BitTorrent, given that this is the standard.

In the original FairTorrent paper, the authors had created an implementation focused on bytes, blocks, and pieces making up whole files: in the context of our simulator, there are only blocks and pieces. To translate the FairTorrent algorithm to the simulator, as many blocks as possible of a given piece will be uploaded when that piece is requested of the FairTorrent and that `Upload` object is chosen. The major parameters fed into the simulator are the following:

1. `numPieces`: The number of pieces in the file to be downloaded.
2. `blocksPerPiece`: The number of blocks needed to be downloaded to finish a whole piece.
3. `minBw`: The minimum number of blocks possible for an agent to send to another agent in a single round.
4. `maxBw`: The maximum number of blocks possible for an agent to send to another agent in a single round.
5. `iters`: The number of iterations for the simulation to be run.
6. `maxRound`: The maximum number of rounds for the simulation to enact before moving on to the following iteration.

7. `Agent, count`: The type of agent (with its corresponding algorithm) and the number of such an agent. This can take multiple different agents and different counts.

Given that different quantities of competing peers yields different results, with higher agent counts yielding greater competition between peers, this should be held constant: the aim of the non-`Seed` peer count is between 9 and 10. In order to prevent our iterations for continuing for an excessive number of rounds, two `Seed`s will be included in each environment. All of the above listed parameters, including the sum of the number of agents will be held constant via the following input:

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

6.2 Results

BitTorrent,9 Freerider,1 As pictured in Figure 3, BitTorrent is robust against Freeriders– positioned against 9 BitTorrents, the Freerider finished its download process last, ending at round 59.1 on average.

Name	Avg. Completion Rounds	Std. Deviation
BitTorrent2	51.5	6.0
BitTorrent0	51.7	5.5
BitTorrent4	51.9	4.8
BitTorrent5	52.1	5.2
BitTorrent7	52.3	5.2
BitTorrent3	52.6	6.1
BitTorrent1	52.7	5.5
BitTorrent6	52.8	6.2
BitTorrent8	53.3	6.2
Freerider0	60.0	3.5

Figure 3: Completion Rounds for BitTorrent,9 Freerider,1

In Figure 4, a visual is provided to highlight the mutual transactions that occur between the BitTorrent, as seen by the darker blocks between them. The entirety of the Freerider row is white, due to its lack of uploading to any other peer; however, due to BitTorrent's optimistic unchokes, the Freerider does end up downloading from the BitTorrents as pictured by the grey squares pictured in the Freerider column, but not as often as the other BitTorrents end up downloading from their fellow BitTorrents, as seen by their darker shade. This difference highlights Freerider's dependence on the `Seed`, which is heavily constrained by its randomized approach and limited bandwidth.

FairTorrent,9 Freerider,1 Figure 5 reveals similar results to those from the environment only featuring BitTorrent and Freerider, showing a strength of the tit-for-tat strategy. However, because of FairTorrent's more generous approach, the average completion rounds are substantially lower than that of the results between BitTorrent and Freerider, given that the last average completion round was 59.1 for BitTorrent and Freerider, and the last average completion round was 56.3 for FairTorrent and Freerider.

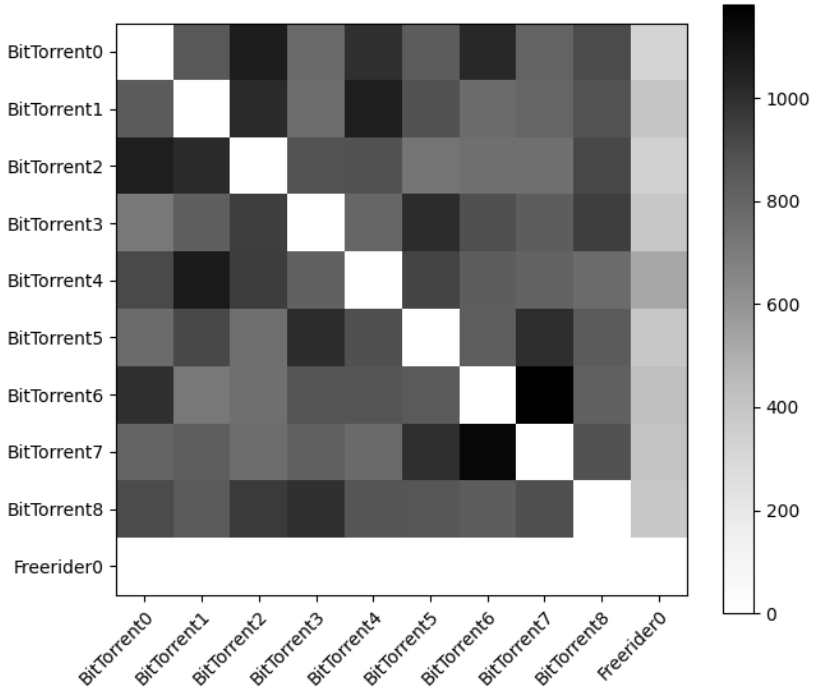

Figure 4: Heatmap generated by BitTorrent,9 Freerider,1

Name	Avg. Completion Rounds	Std. Deviation
FairTorrent5	47.0	5.4
FairTorrent2	47.2	5.6
FairTorrent6	47.8	6.0
FairTorrent1	48.1	6.0
FairTorrent8	48.2	6.5
FairTorrent3	48.6	7.1
FairTorrent7	48.7	7.6
FairTorrent0	48.9	8.0
FairTorrent4	49.3	6.1
Freerider0	56.3	2.6

Figure 5: Completion Rounds for FairTorrent,9 Freerider,1

Maintaining that train of thought, FairTorrent's altruism is also shown in the heatmap for the environment, pictured in Figure 6. Because FairTorrent does always upload to at least one of the requests that is submitted to it, it is more inclined to upload to a Freerider than BitTorrent because of it's unability to actively choke its peers. This is present in the darker shade present from the FairTorrents to Freerider0 column (in comparison to the column present in the BitTorrent and Freerider environments). Furthermore, because of this lack of active choking and the longer-term approach to tit-for-tat, the unchokes pictured in the figure from FairTorrents to other FairTorrents are not as even or as consistent as the spread between BitTorrents and other BitTorrents seen in Figure 4. This is especially notable in the relationship between FairTorrent1 and FairTorrent2, being substantially lighter than the surrounding squares.

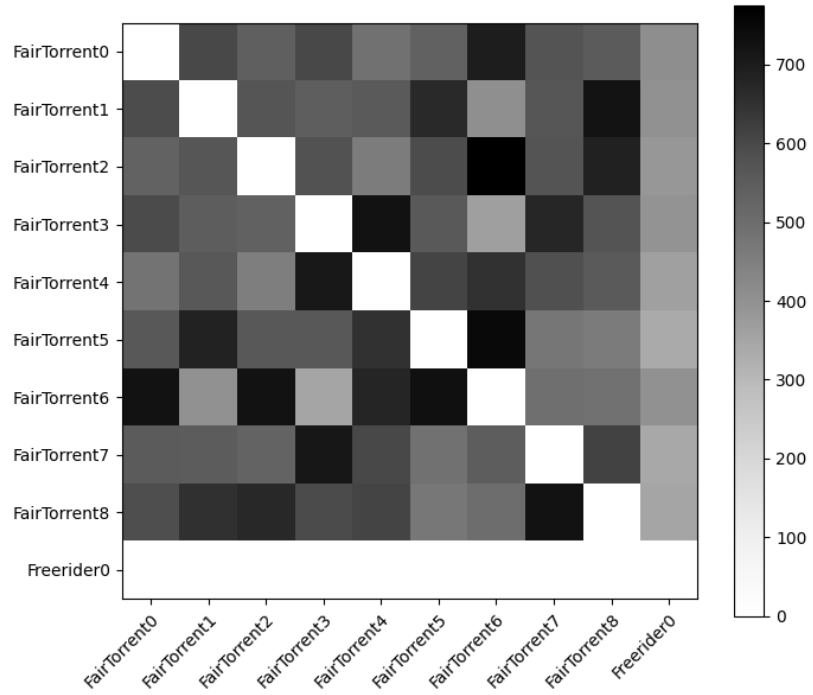

Figure 6: Heatmap generated by FairTorrent,9 Freerider,1

FairTorrent,5 BitTorrent,5 FairTorrent's altruism, in comparison to BitTorrent's altruism, which manifests as the optimistic unchokes and even division of bandwidth when it comes to uploads as described in Section 4.1, appears to be much more efficient in regards to download completion time, as seen in Figure 7. What is also of note is that the table reveals that the standard deviation appears to be much greater for BitTorrent than FairTorrent, when the two are directly put up against each other.

When taking a look at the heatmap of unchokes, featured in Figure 8, it appears that the chart is split up into four quadrants: while the two different types of algorithms do exchange data with each other, it appears to be a much lesser degree than with agents that possess an algorithm of the same type. Like seen earlier, the quadrant of BitTorrents appear to be much more consistent with collaborating with all of the agents of their type, while FairTorrents also do aggressively collaborate

Name	Avg. Completion Rounds	Std. Deviation
FairTorrent0	45.4	5.0
FairTorrent4	45.7	4.1
FairTorrent1	46.1	4.6
FairTorrent2	46.5	5.8
FairTorrent3	46.6	4.2
BitTorrent2	48.6	8.5
BitTorrent1	48.8	8.1
BitTorrent4	50.2	7.0
BitTorrent3	50.8	6.4
BitTorrent0	50.8	7.9

Figure 7: Completion Rounds for FairTorrent,5 BitTorrent,5

with each other, although they do appear to choose random favorites whom they unchoke more often than others.

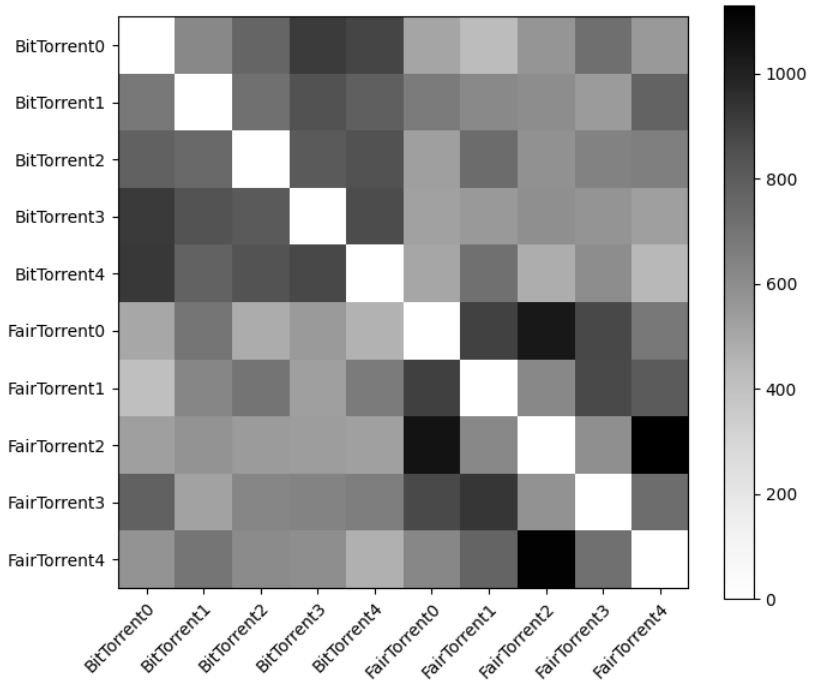

Figure 8: Heatmap generated by FairTorrent,5 BitTorrent,5

Due to this similar behavior, but Figure 7 still revealing that FairTorrent relatively consistently beat out BitTorrent in terms of completion time, it was shown that the upload protocol itself was notable: not only in how the peer to which to upload to was chosen, but also in the quantity of blocks that were chosen to be uploaded. This is revealed in Figure 9; while it looks deeply similar to Figure 8 in terms of shape, with both the top-left and bottom-right quadrant being darker than the other two, in Figure 9, the relative darkness of the BitTorrent to BitTorrent squares is much

lighter than the FairTorrent to FairTorrent squares. Furthermore, it appears that FairTorrent was indeed more generous than BitTorrent in terms of cross-protocol uploads, given that it appears that the FairTorrent to BitTorrent quadrant appears to be on average darker than the BitTorrent to FairTorrent.

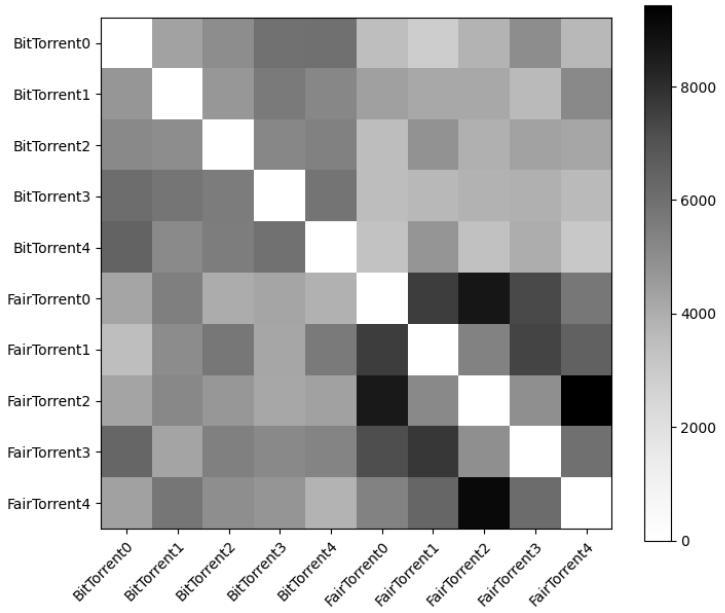

Figure 9: Heatmap for Blocks Exchanged by FairTorrent,5 BitTorrent,5

This reveals that FairTorrent's protocol regarding uploads allows for more focused collaboration, supporting the theoretical premise of Sherman, Nieh, and Stein's paper that FairTorrent is more effective in terms of download speeds than BitTorrent due to the ability to more efficiently utilize bandwidth space.[4] From a theoretical standpoint it appears that by uploading entire pieces at a time when possible, FairTorrent aggressively incentivizes further cooperation with other FairTorrents.

FairTorrent's Approach to Collaboration Based off the preceding results, it appears that the following theoretical analysis appears reasonable: FairTorrent's upload process differs from that of BitTorrent by uploading as much of a given piece as possible; if there is leftover bandwidth, then a following piece will begin to be uploaded for another peer (as described in Section 4.2). When a FairTorrent A is working with only other FairTorrents, this yields more mutual collaboration by adjusting the deficit value of the peer with the highest deficit value (call this peer B); this is through uploading entire pieces consisting of a given set of blocks (or the maximum number of blocks given a set bandwidth– whichever is smaller), the deficit value from the perspective of peer B regarding peer A as a result will also decrease, resulting in A moving down the deficit list for peer B, making A more likely to be the recipient of an upload when they put in a request in a future round.

7 AngwyTorrent

7.1 Rationale

Based off the results in Section 6.2, FairTorrent appears to be a more efficient method of implementing tit-for-tat in terms of working in swarms and functioning against Freeriders. However, there is a question regarding whether or not there is a simpler way to break the robustness of tit-for-tat. For the final section of this paper, a strategy will be implemented that will intentionally attempt to mirror the tit-for-tat in its opposite. This method will be theoretically touched on and then empirically explored, and its related automata is pictured as the following:

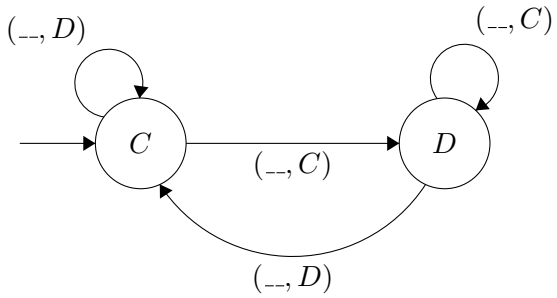

Figure 10: Automata for AngwyTorrent

This protocol will be called AngwyTorrent due to its, practically speaking, irrational approach. When a peer cooperates, the AngwyTorrent will transition to dissenting, if it isn't already– and when a peer dissents, the AngwyTorrent will transition to cooperating if it isn't already. Figure 10 features an automata that is identical to that of tit-for-tat, as articulated in Section 2 of this paper, except each of the edges has the opposite action listed: when tit-for-tat features an edge that is $(\_, D)$, then the automata for AngwyTorrent features the same edge with $(\_, C)$; and when tit-for-tat features an edge that is $(\_, C)$, then the automata for AngwyTorrent features the same edge with $(\_, D)$. Both automatae start with cooperation; empirically, this due to the fact that when the deficit values for all of the peers are the same from the perspective of a given agent, then it does benefit to upload to anybody who requests it, because then bandwidth is more efficiently utilized, and theoretically, it ensures that tit-for-tat will cooperate in the following round in the context of BitTorrent and will cooperate in rounds far into the future in the context of FairTorrent.

When running both automatae, assuming that both machines start at C, then the following progression should occur and then repeat itself once the process is completed, where the first item in the touple is the AngwyTorrent and the second item is a tit-for-tat agent:

1. (C, C)
2. (D, C)
3. (D, D)
4. (C, D)

Figure 1 in Section 2 features a pay-off matrix for a prisoner's dilemma, which is being used as a method of articulating a P2P network game in the context of this paper. Given the above possibilities, according to the pay-off matrix, assuming that either of the two players may choose

to dissent as a result of focusing on an alternate peer with whom to exchange, in half of these scenarios, the AngwyTorrent will be at an advantage: in the remaining scenarios, the utility of the two players should be even, meaning that the tit-for-tat's strategy will only ever yield as much utility as the AngwyTorrent's strategy theoretically. Given this strategy, it would appear reasonable to conclude that when implemented, AngwyTorrent would yield quicker download times than either of the tit-for-tat strategies, given enough iterations.

Kill them with Kindness? AngwyTorrent still resembles tit-for-tat, technically: given a change in action from the peer, AngwyTorrent retaliates accordingly. However, it punishes cooperation and rewards dissent, and through this, exploits actual tit-for-tat strategies from a theoretical standpoint. AngwyTorrent does not maximize the overall good of the environment by focusing on increasing the quantity of (C, C) events, which mutual tit-for-tat strategies like BitTorrent attempt to maintain with other agents of the same type; instead, AngwyTorrent attempts to maximize the amount of time that AngwyTorrent has benefited more than the opposing tit-for-tat strategy, and given that empirically, these protocols are functioning in an environment with more than two players and a limited quantity of bandwidth, it is inevitable that the process of the two communicating automatae by which the AngwyTorrent and tit-for-tat protocol are playing will be interrupted and AngwyTorrent will be forced to play D even when it isn't appropriate according to the automata.

When such a process is interrupted, either the following round will return to stage 2– that being (D, C)– or stage 3– that being (D, D)– depending on which moment the process was in. From there, the process may resume to the following stage depending on when the AngwyTorrent returns to communicating with the peer, yielding greater benefit for the AngwyTorrent. Through this reasoning, AngwyTorrent should end up often completing the download process before any tit-for-tat strategy.

7.2 Implementation

The implementation of AngwyTorrent is identical to that of FairTorrent, with the exception of the upload function: rather than sorting the list by the lowest deficit, AngwyTorrent sorts it by the highest deficit, and then chooses to upload to the peers that way, traversing down the list.

7.3 Results

BitTorrent vs. AngwyTorrent Figure 11 reveals that AngwyTorrent does outperform BitTorrent. This is corresponds with the results found in Section 6.2; FairTorrent was quicker to finish than BitTorrent due to its upload process, which is also featured in AngwyTorrent. It also appears that the margins by which AngwyTorrent finishes first appears to be mildly greater than the margins by which FairTorrent beat BitTorrent. In Figure 7, the half range of the avg. completion rounds for the entirety of the environment is calculated as the following: $(50.8 - 45.4)/2 = 2.7$, in contrast to the results in Figure 11, where the half range is $(52.3 - 44.9)/2 = 3.7$. This greater half range for the average completion rounds between the two environment is marginal but may present an occasional strength of AngwyTorrent against FairTorrent, which has theoretical support described in Section 7.1.

The heatmap featured in Figure 12, similar to the relationship of FairTorrents to other FairTorrents in Figure 8, the pairs which feature higher quantities of unchokes appear to be randomly assigned

Name	Avg. Completion Rounds	Std. Deviation
AngwyTorrent1	44.9	5.9
AngwyTorrent0	45.3	6.4
AngwyTorrent2	45.3	5.0
AngwyTorrent4	45.4	6.4
AngwyTorrent3	46.8	6.2
BitTorrent2	48.4	5.1
BitTorrent0	48.6	5.9
BitTorrent3	50.7	6.0
BitTorrent4	51.5	6.0
BitTorrent1	52.3	7.3

Figure 11: Completion Rounds for AngwyTorrent,5 BitTorrent,5

between pairs of AngwyTorrents. It does not appear that AngwyTorrent's presence has not substantially changed the degree of inter-protocol exchanges.

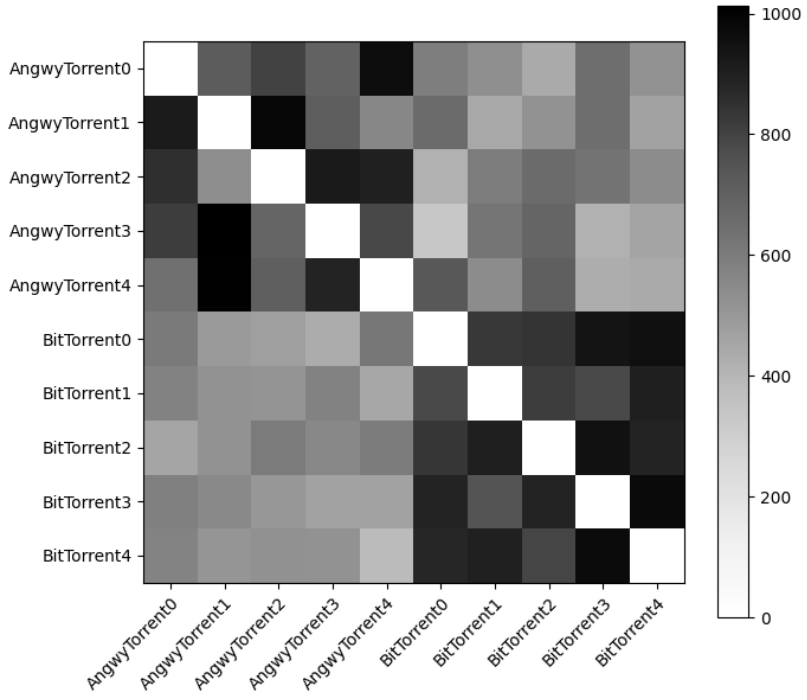

Figure 12: Heatmap generated by AngwyTorrent,5 BitTorrent,5

FairTorrent vs. AngwyTorrent In Figure 13, it appears that AngwyTorrent and FairTorrent are much closer competition than AngwyTorrent and BitTorrent, fitting the theoretical explanations regarding the strengths of AngwyTorrent; the presence of two `Seed`s results in a possible dependency of both types of protocol on the `Seed` which may be the final dictator for the rate of completion,

Name	Avg. Completion Rounds	Std. Deviation
AngwyTorrent0	44.2	5.9
AngwyTorrent3	44.4	6.4
AngwyTorrent1	44.9	5.0
AngwyTorrent4	44.9	6.5
FairTorrent2	44.9	4.4
FairTorrent3	45.8	6.9
AngwyTorrent2	45.8	6.9
FairTorrent1	46.3	4.6
FairTorrent4	46.6	4.9
FairTorrent0	46.8	5.7

Figure 13: Completion Rounds for AngwyTorrent,5 FairTorrent,5

and due to the `Seed`'s random approach may result in certain anomalies. This is clearly seen in AngwyTorrent2 being the only AngwyTorrent that did not complete its download before any of the other FairTorrents on average, being surpassed in the figure by FairTorrent2 and FairTorrent3, albeit by very slim margins.

Furthermore, in Figure 14, there is a similar kind of randomness present in the AngwyTorrent to AngwyTorrent quadrant as there is in the the FairTorrent to FairTorrent quadrant. The two other quadrants appear to be relatively lighter than the cross-protocol quadrants present in the other figures in Section 6.2 as well as Figure 12 specifically, revealing that AngwyTorrent's protocol stratifies the connections between itself and FairTorrent more than AngwyTorrent does with BitTorrent.

8 Conclusion

It appears that AngwyTorrent playing directly against tit-for-tat does indeed typically yield more efficient download times for AngwyTorrent, given that they are in a context where all of the players feature some sort of tit-for-tat strategy. In contrast to BitTorrent and FairTorrent, however, under no conditions would AngwyTorrent be able to prevent the success of Freeriders, given that Freeriders never upload, meaning that Freeriders will always have the highest deficit and will always recieve downloads from AngwyTorrent first. However, it's possible that by removing the Freerider from the environment as quickly as possible, it will allow the remaining nodes to finish their download processes more quickly, instead of forcing the Freerider to depend purely on the `Seed` for its downloads. This could theoretically yield a lower half-range for the average completion time for the given environment, potentially warranting future study. Another area of future work revolves around much more specifically quantifying the quantity of inter-protocol exchanges/unchokes and the quantity of intra-protocol exchanges/unchokes. While the heatmaps were helpful in visualizing the data, by quantifying these values, the results and analysis regarding these sorts of communications will be much more concrete and replaceable.

A strength of FairTorrent established is its ability to upload entire pieces at a time: the bandwidths assigned to each of the peers in this experiment was 24, while each pieces was made up of 16 blocks. This allowed for FairTorrent to send a whole piece of the file during every round

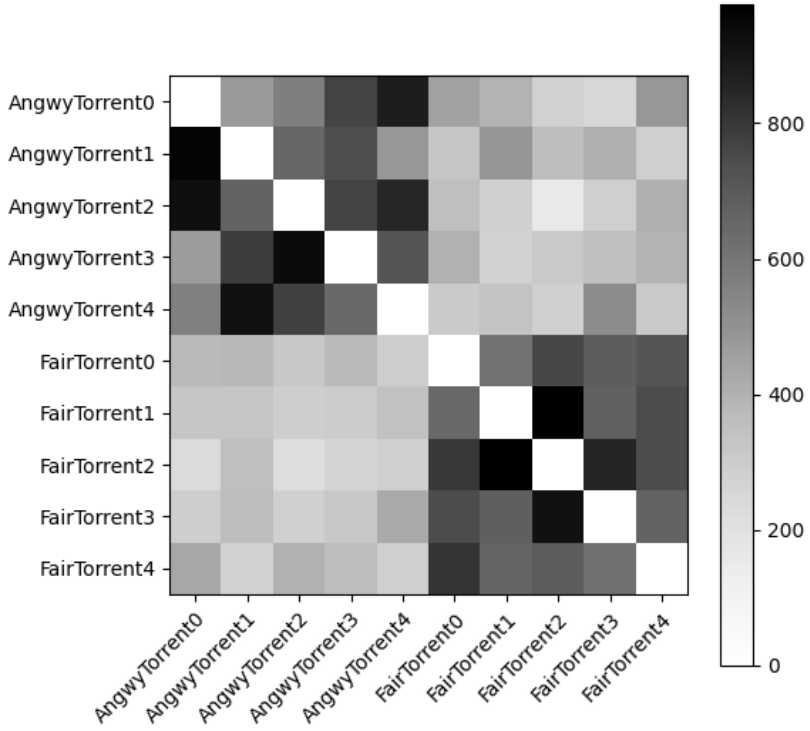

Figure 14: Heatmap generated by AngwyTorrent,5 FairTorrent,5

and was a notable strength of it– an aspect worth exploring as a result is the outcome of having pieces that are greater in size than the bandwidth assigned. This may decrease the significance of FairTorrent's strength as a protocol and may result in the distribution between unchokes to be different. Another thing worth exploring is running the same experiments with only a single `Seed`, rather than two. This may result in the `Seed` being less of a randomized crutch for the nodes such that the power of the AngwyTorrent is highlighted.

References

[1] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of PeertoPeer systems*, volume 6, 2003.

[2] Umair Waheed Khan and Umar Saif. Bittorrent for the less privileged. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, New York, NY, USA, 2011. Association for Computing Machinery.

[3] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in bittorrent. In *Proc. of NSDI*, volume 7, 2007.

[4] Alex Sherman, Jason Nieh, and Clifford Stein. Fairtorrent: Bringing fairness to peer-to-peer systems. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, page 133–144, New York, NY, USA, 2009. Association for Computing Machinery.