CSC523: Analysis of the P2P BitTorrent Protocol

Abram Hindle 0020755

April 16, 2004

1 Introduction

In this paper we look closely at the BitTorrent P2P protocol. We extract problems that have already been studied from the protocol and discuss those problems. Some problems we subject to further analysis while creating new ones to solve.

BitTorrent's range of problems crosses many domains, from computer networks, to sociology to economics, to computer security issues. BitTorrent covers a wide gambit of problems and thus many will be discussed. There is also a literature of the general state of P2P in relation to BitTorrent.

Problems examined consist of Tit For Tat strategy in Iterated Prisoners Dilemma, Various P2P Questions, Byzantine Generals Problem, and Hashing.

1.1 What is BitTorrent

BitTorrent is a P2P protocol meant for distributing files. The purpose behind BitTorrent is to reduce the bandwidth load for the peer (the seeder) initially sharing the file . Peers who download from the seeding peer join the network and share their blocks of the file with other clients. Each file is split into blocks so a seeder can distribute blocks among the downloading peers such that peers can download the blocks off other peers. Thus a downloader effectively becomes an uploader. As more peers join they connect to the downloading peers and trade file blocks from them.

To promote sharing of bandwidth a Tit For Tat algorithm is implemented on each peer. This suggests that a peer must send data to another peer if it expects the other peer to send data back. Thus to successfully download from a BitTorrent network one has to allocate some of their upstream bandwidth to the network otherwise suffer very slow transfers.

BitTorrent is interesting as it is currently used by many users to distribute large files. Originally used to distribute legal high quality bootleg recordings of live concerts, BitTorrent is now very popular with those who trade television, movies, arcade games, comic books and music illegally. BitTorrent is also used heavily in the Linux community to distribute like files such as CD images. The popularity of BitTorrent is likely due to the control the seed has over the network as well as the ability for a seed to distribute a file's whole size once while. A seeder can post a torrent file on their webserver and say "If you want this file jump on to this torrent". A user downloads the torrent and BitTorrent downloads the file. The focus of effort and lack of "always-on" P2P sharing software makes it especially useful in small community like those based on messageboards. There are many communities which trade television shows and comics through semi-private webboards using BitTorrent. Due to BitTorrent's tit for tat bandwidth sharing policies often the seeder can send out a file once and have it distributed to many due to the P2P network made around that file. Thus effectively by uploading the file once, its persistence within the network depends on how long other hosts stay on the network.

2 Terms

- Torrent - A file which provides a URL to the tracker as well contains a list of SHA1 hashes for the data being transfered. This is so that the hashes in the Torrent can be used to verify if the blocks received are valid or not.
- Tracker - A middleman who informs the peers of all the other peers in the network.
- Peer - A client to the network dedicated to a torrent.
- Seeder - A Peer who has all the blocks in a torrent.
- Choked - A connection is choked if not file data is passed through it. Control data may flow but the transmission of actual blocks will not.
- Interest - indicates whether a peer has blocks which other peers want.
- Snubbed - A peer acting poorly - not uploading - or sending bad control messages, usually disconnected or ignored.

3 Literature Review

I read various papers on issues relating P2P. Some related to BitTorrent, while other related slightly to analysis using techniques such as Chernof Bounds or iterated prisoners dilemma. Here are some reviews of some of them.

"Analyzing peer-to-peer traffic across large networks" by Sen and Wang [10] discussed how P2P traffic actually looked on a large network. They analyzed the network traffic of an ISP (probably a AT&T owned ISP) and concluded the results. They took statistics about packet sources and destinations and reasoned that many of the current p2p network available today can be taken down by the removal of a few important nodes. They also were able to notice the difference p2p users using networks that used supernodes and those which didn't. They observed the phenomena of less than 10% of the hosts being responsible for more than 90% of the traffic and content on the file based P2P networks.

Some of my research went into how agents act. "Emotional Pathfinding", [11] by Donaldson, Park and Lin described agents who prioritize goals using emotions. Emotions were abstracted away to the idea of dominant priorities that can increase or decrease in importance and override other goals. This is to avoid certain erratic and fast acting

behaviors which can be detrimental to the agent or the system. This paper provided a good basis for what a computer scientist means by the term "emotional". It's almost like smoothing the transitions between priorities. "A Framework for the simulation of Agents with Emotions", [3] by Bazzan and Bordini shared the same idea of emotions from the previous paper and applied it to the prisoner's dilemma. Unfortunately they only simulated the results. In their trials of iterated prisoners dilemma, the emotional agents usually defeated the always defect and always cooperate agents. The emotional agents were not tested against tit for tat agents because the researchers thought that defeated the purpose and the results would have been predictable. Tit for tat likely would have been dominant against an emotional agent. The emotional agents are relevant to BitTorrent as BitTorrent acts similar to an optimistic tit for tat algorithm.

"Notions of reputation in multi-agents systems: a review", by Mui, Mohtashemi, and Halberstadt [8] discussed the state of reputation in agent based systems. Discussed were different ways for measuring, observing and inferring reputation. Examples of real world systems were given such as eBay buyer/seller feedback. They then tested many of these systems for reputation in Iterated Prisoners Dilemma. The conclusion seemed to be that reputation based on what a trusted group of peers had already concluded about another peer resulted in a better performing agents in Iterated Prisoners Dilemma.

"Towards a Pareto-optimal solution in general-sum games", by Sen, Airiau and Mukherjee [9] discuss techniques in which agents can learn to not necessarily cooperate but to optimize their strategy in a Markov game (like iterated prisoners dilemma) in order to achieve Pareto-optimal solutions that beat Nash Equilibrium solutions. Concepts such as desired states versus greedy states are brought up. A desired state is where both agents made a choice which results in the greatest payoff for both of them together. A greedy state is one which both agents choose that state because they are given safe but high payoffs. This was to also use learning to determining whether given the opponent it was feasible to use a greedy or desirable strategy. This is related to BitTorrent because BitTorrent often takes optimistic approaches to game theory in the hopes of gaining faster, more cooperative peers.

"The Byzantine Generals Problem" [5] by Lamport, Shostak and Pease introduced the Byzantine General's Problem to computer science. The problem was proposed and analyzed in the paper, the problem is about how a group of traitors can confuse messages and cause miscommunication, cause disagreement or agreement based on their bias. These traitors could be working together to subvert the integrity of the final decision. The paper covers many different aspects of the problem from communication network disruption to bad commanders.

"Scalable Byzantine Agreement" [6] by Lewis and Saia discussed an algorithm to solve the Byzantine Agreement using an randomized algorithm. It uses randomness and probability on the Byzantine General's Problem of Byzantine Agreement such that using $O(logn)$ messages sent each for each of n peers ($O(logn)$ rounds) the the problem of reaching a trustable agreement is solved. They make an interesting but valid claim that "In fact, for a p2p system to be scalable we generally require all resource costs per peer to be polylogarithmic in the number of peers".

"FARSITE: Federated, Available and Reliable Storage for an Incompletely Trusted Environment" , by many authors at Microsoft Research [2], describes a distributed file system like much like NFS except based around the idea that not all the peers who will store and keep information can be trusted. This decentralized storage technique is to take advantage of a large companies unused storage space found on their employee's computers. An office might have 40 computers each with 20 Gigabytes of free space. Effectively there is 800 gigabytes of unused space that could be used in FARSITE file system. FARSITE is interesting because it supposedly protects itself against byzantine faults. Unfortunately the paper doesn't really go into what they do to protect themselves, they just make reference to the name of fault. This is what one would expect from a paper from a industry research group.

4 Problems

There are many problems surrounding P2P protocols and the BitTorrent Protocol itself.

4.1 Block Distribution Problem

Block Distribution is a concern for BitTorrent. Peers join and exit the network all the time. If a seeder uploads to a peer that leaves, many of those blocks are lost. Thus a host is often obligated to distribute blocks amongst peers such that when a peer leaves a large subset of blocks is not missing.

BitTorrent peers usually request random blocks initially then request the rarest blocks from that point on.

4.1.1 Random Block

The connecting peer doesn't know much about the network it connects to. If it choses the rarest block first, the peer could be slowed since this block could be rare due to a very slow host. If the peer requests a block the seeder has already sent (possibly the rarest block) the seeder is less likely to send that block to the peer as it's already been sent once.

Thus by requesting a random block first the peer has a better chance at receiving a full block from a good source

than asking for a rare block.

Random block is most useful when a client doesn't know the layout of the network well yet. Although it seems somewhat strange. Wouldn't it be better to download the first block from the network as it'd be guaranteed to be there? Potential issues with using one static block would be that no one on the network wants that block thus you rely on the optimistic parts of the tit for tat algorithm. By requesting a totally random block you slowly get a block which is likely to be rare to other peers such that you can participate in the network effectively.

4.1.2 Rarest Block

Rarest Blocks are requested by the peers. By rarest we mean the block that the fewest peers have. By requesting rarest blocks, peers try to keep all the blocks available to the network. This reduces the reliance on one peer to host one block and provides redundancy as the weakest parts are backed up first.

I will demonstrate how using the rarest block first algorithm we will still get all the blocks in the network. Let there be n peers. Let there be a file of m chunks. For each m chunks assume at least 1 peer has that chunk. P_m indicates how many peers have chunk m. The rarest block is the $\{x|x \in 1..mP_x = min(P_1, ..P_m)\}$ Everyone in the network has all blocks when $P_1 = P_2 = .. = P_m = n$. Thus if a peer doesn't have a block and the block is available in the network then that block will be the rarest or eventually become the rarest. The the peer will get that block and if $P_1 = .. = P_m = n$ then everyone has downloaded all the necessary blocks.

Given n peers and 1 seeder. How many times does the seeder have to upload to send the whole file of m parts to the n peers assuming each peer drops off once they have all the pieces.

So our upper bound is naively $n * m$ parts uploaded.

Given or Assumed:

Lets assume each peer is connected to all other peers.

$n < m$

Each peer is connected to all other peers.

Each peer is connected to the seeder

Assume no preferences or priority to uploaders

Assume each turn every peer can upload 1 chunk to 1 peer and receive 0 or more chunks from another peer

Lets assume each peer has 1 chunk and the seeder has all the chunks.

Lets assume a peer randomly chooses another peer to send to

Chance a peer is not chosen in a round, $((n-2)/(n-1))^n$

What is the expected number of peers that chose any given peer?

The probability of being chosen is $p = 1/(n-1)$

$E[X_i] = n * p = n/(n-1)$, thus it expected that $E[X_i] = n/(n-1)$ where X_i is a random var indicating how many peers chose peer i.

We want to know how many rounds before a peer has all the blocks.

Based on coupon collector problem [7] if we are offered one random block per turn, $E[X] = 2nlnn + c$ where c is a constant.

This would only be true if the distribution was uniform per turn of a block being offered.

Lets skip to Turn m where the seeder has uploaded all the blocks to network.

Assuming a uniform distribution, X_i is a r.v. which indicates how many blocks peer i has. $E[X_i] = m/n$. Thus each peer has $m - E[X_i]$ blocks left.

If we assume a system where everyone uploads randomly to everyone else. This is basically the coupon collector problem. Each round we get a block randomly from someone. Lets assume that our current collection of blocks is unique to us thus the likely hood of getting one of the initial $E[X_i]$ blocks back is initially zero and slowly starts to grow to be bounded by $E[X_i]/m = 1/n$. So we'll ignore the duplicates thing. Taking the previous results from coupon collector $E[X] = 2nlnn + c$ we now apply it to our current situation where X is a r.v. indicating how many rounds before we have all the blocks if we're sent a block randomly each round. $E[X] = 2(m-m/n)ln(m-m/n)+c$

So the expected number of turns til everyone has their blocks should be $m + 2(m - m/n)ln(m - m/n) + c$.

Let's compare our model's results with the real world. These results are from the empirical section 6.2, see there for any constraints on the experimental results. See figure 1 for the data and the graph of the data in figure 2. The comparison between the experimental results and the model aren't quite valid for 2 reasons. In the experiment the peers drop out when they are finished and the network had shared but limited bandwidth.

4.2 Game Theory

Game Theory applies directly to BitTorrent as aspects of game theory help determine fair strategies that promote bandwidth sharing and the distribution of load among peers.

4.2.1 Pareto Efficiency

A Game that is Pareto efficient if there no way someone is better off without making someone worse off [1]. In BitTorrent this is used to spur peers to look for better peers or at least be fair and communicate with many peers.

file chunks	experimental	n=4 log(10)	n=4 log(2)		n=5 log(10)	n=5 log(2)	n=5 log(exp(1))
1.00	17.86	0.81	0.38	0.57	0.84	0.48	0.64
2.00	22.05	2.53	3.75	3.22	2.65	4.17	3.50
4.00	30.74	6.86	13.51	10.59	7.23	14.74	11.44
8.00	48.17	17.34	39.02	29.50	18.32	42.28	31.76
16.00	81.60	41.90	102.04	75.64	44.34	110.16	81.27
32.00	154.20	98.25	252.08	184.55	104.10	271.52	198.02
64.00	303.65	225.40	600.16	435.64	239.03	645.43	467.02
128.00	639.06	508.60	1392.31	1004.35	539.71	1495.67	1076.00
256.00	1309.33	1132.79	3168.63	2274.88	1202.72	3400.94	2435.91

Figure 1. Experimental Results Versus Model Results File Chunks Versus Turns/Time

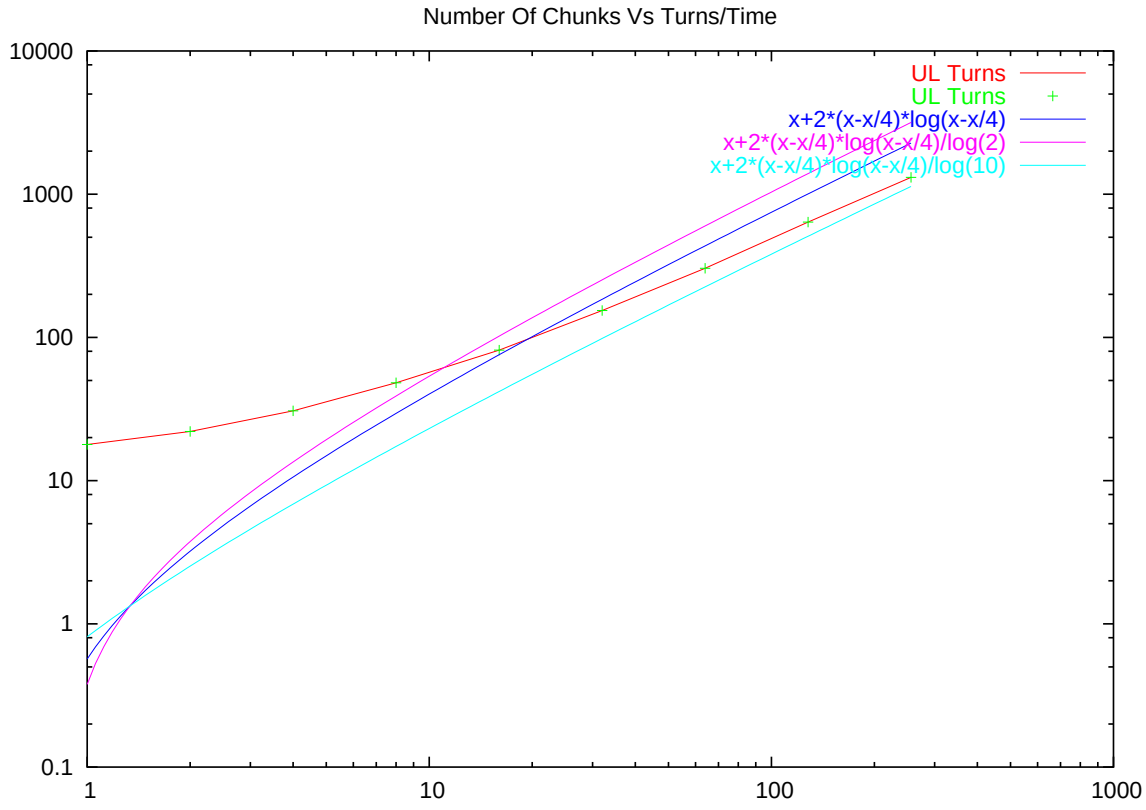

Figure 2. Experimental Results Versus Model Results File Chunks Versus Turns/Time

> In computer science terms, seeking Pareto efficiency is a local optimization algorithm in which pairs of counterparties see if they can improve their lot together, and such algorithms tend to lead to global optima. Specifically, if two peers are both getting poor reciprocation for for some of the upload they are providing, they can often start uploading to each other instead and both get a better download rate than they had before.[4]

Effectively BitTorrent is designed to promote the sharing of bandwidth in order to improve transfer rates between peers.

4.2.2 Tit For Tat

Tit For Tat is a strategy for uploading and downloading between peers. There are pessimistic and optimistic tit for tat algorithms.

Tit For Tat is a strategy used in game theory problems such as prisoner dilemma where you take the strategy of your opponent. If they cooperate, you cooperate next turn. They don't cooperate; you don't cooperate.

How would a random strategy fair against tit for tat? By fair we mean minimize the time it takes to download while minimizing the amount of data uploaded. A strategy that doesn't upload much downloads a lot is a good strategy.

Lets create an small game.

Lets define the tit for tat strategy as for the first round we always upload. Given a round k where $k > 1$ and $k \in Z$ and $b_{k-1} = \{1$ if B uploaded during round $k-1$ or 0 if B didn't $\}$

Tit for tat strategy uploads if b_{k-1} was 1. Otherwise it doesn't upload.

How does tit for tat fare against a random opponent? We'll define a random opponent as an opponent who given a probability p uploads to A.

Thus the expected number of times a random opponent B uploads during a n round game is $n * p$. An optimistic tit for tat would result in an expected number of uploads either being $n * p$ uploads or $n * p + 1$ uploads during the game of n rounds, $E[X]$ or $E[X] + 1$.

Event 1 B doesn't upload at the start, resulting in +1 more uploads. Thus Event 1 occurs with a probability of $(1 - p)$.

Event 2 B does upload a the start resulting in 0 more uploads. Thus Event 2 occurs with a probability of p.

Thus for A the $E[X]$ where X is the number of uploads A makes and Y is the number of uploads B makes is $p * E[Y] + (1-p)(E[Y]+1)$ which is $p*n*p+(1-p)(n*p+1)p*n*p+n*p+1-n*p*p+-p = n*p-p+1$. Thus $E[X] = n * p - p + 1$

This is a naive approach to game theory with BitTorrent.

4.3 Related Problems

4.3.1 Byzantine Generals Problem

The Byzantine Generals Problem is related to BitTorrent more so as a warning against sabotage on the the BitTorrent network. Sabotage could come from copyright holders to Internet vigilantes to hackers.

This relates to BitTorrent. How does BitTorrent defend against colluding peers that seek to subvert the network? An area in BitTorrent where this could be used in detecting if a peer or a group of peers is lying about their upload / download statistics to the tracker. If everyone voted and agreed what one client uploaded that might work out quite well. Especially if it only takes $O(logn)$ as suggested by Lewis and Saia [6].

In BitTorrent If a peer detects invalid data from another peer such as damaged datastructures or improper field lengths, it automatically disconnects that peer. If a peer sends invalid data to another peer, this will be noticed as the SHA1 hash from that chunk will not match.

5 Hashing

SHA1 hashes are used by BitTorrent on chunks of the file. The size of the chunks range from 64KB to 1024KB. The chunks are always of sizes 2^n where n is greater than or equal to 16.

SHA1 hashes are 160-bit, thus naively the likelihood of any two strings having a matching checksum is 2^{-160}. As far as I can tell no one has found a SHA1 collision yet.

Thus if a peer's packets get modified or garbled or it's original file is not complete or is corrupted the other peer will know. If a peer keeps sending bad data it will be snubbed and ignored.

6 Further Analysis

6.1 BitTorrent Code

Bram Cohen's code for BitTorrent is well written. It lacks commenting (per each file there are 2 lines of comments at the top of the file). The language BitTorrent was written in is Python. A clean cut scripting language which has some nice array manipulation operators. Python is virtually equivalent to Perl.

The Choker is a rather interesting module. Every 30 seconds a round robin scheduler runs and checks for connections with are choked but are interested in participating. This scheduler rotates the connections array until the first choked but interested connection appears.

At least every 10 seconds the connections are reprioritized using the rechoke method.

_	B doesn't upload to A	B uploads to A
A doesn't upload to B	(0,0)	(1,0)
A uploads to B	(0,1)	(1,1)

Table 1. In relation to A's actions (A receives,A sends)

The algorithm for rechoking is such:

- create a list called 'preferred' of connections are not snubbed but are interested
- reverse sort 'preferred' by the upload rate (so the largest uploading connections appear first)
- cut the the tail off the list to reduce the size to max_uploads in size
- for all connections
  - unchoke the connection if it is in the list 'preferred'
  - otherwise unchoke the connection if we have less unchoked and interested connections than min_uploads or if we haven't found a interested connection yet.
    - If we find an interested connection we increment our upload count . Thus we will keep unchoking until a min_uploads number of connections have been unchoked.
  - otherwise choke the connection.

Lots of the BitTorrent code relies on randomization to provide fair and optimistic strategies. In the choker when one adds a connection it is added randomly to the connection list (with a slight bias for the front of the list).

Lets take a look at a tiny sample of the code in figure 3.

This method is called when a connection is made and the connection is added to the Choker (who chokes and unchokes connections, as well as prioritizes them). It takes in 3 arguments, the object itself, the connection and the priority p. If p is not set, the connection is randomly placed inside the connection list associated with the object.

Randrange chooses a values from the range $[-2, len(self.connections) + 1)$ thus the range is $[-2, len(self.connections)]$. Using that value the connection is inserted into the list. In the forth line we see, if the value for p is less than 0 we use 0 instead. Thus $\frac{3}{n+3}$ is probability that an element will be put at the head of the list, where n was the size of the list originally. Where as for all other positions it is $\frac{1}{n+3}$, thus $1 = \frac{3}{n+3} + \frac{n}{n+3}$.

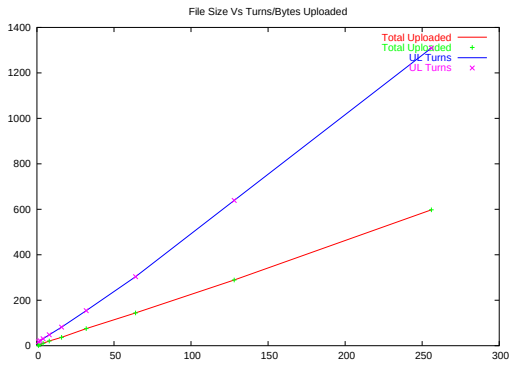

Figure 5. Given files of size 2^0 to 2^8 megabytes in size we see how much the seeder uploads to 4 other peers on a shared network

6.2 Empirical

A test framework was created to enable the easy creation and execution of tests using BitTorrent. A network of 5 computers was setup and linked together using a 10/100 switch. The 10/100 switch caused difficulties as it allowed some hosts to access 10 times the bandwidth than other hosts. Instead a 10 MBit hub was substituted.

Given a little bit more time I would be able to do more effective testing. Such as ones which don't require the peers to exit of finishing.

In figure 4 an example run of the network is given. The seeder is the first peer . The figure describes the upload and download rates in bytes per second of each peer in the network. Each peer is set to disconnect once they have the full file. These experiments take place on one Ethernet network, thus the bandwidth is shared bandwidth. When more peers leave, they stop using the network bandwidth thus the seeder uploads faster. A problem with the experiment is that the machines running the BitTorrent software are heterogeneous. They differ in just about every way other than software. The software is exactly the same except for the drivers that the kernel loads.

In figure 5 we graph the linear relation between how much the seeder uploads and how big the files are.

```
def connection_made(self, connection, p = None):
    if p is None:
        p = randrange(-2, len(self.connections) + 1)
    self.connections.insert(max(p, 0), connection)
    self._rechoke()
```

Figure 3. Python Code that assigns priority to new connections

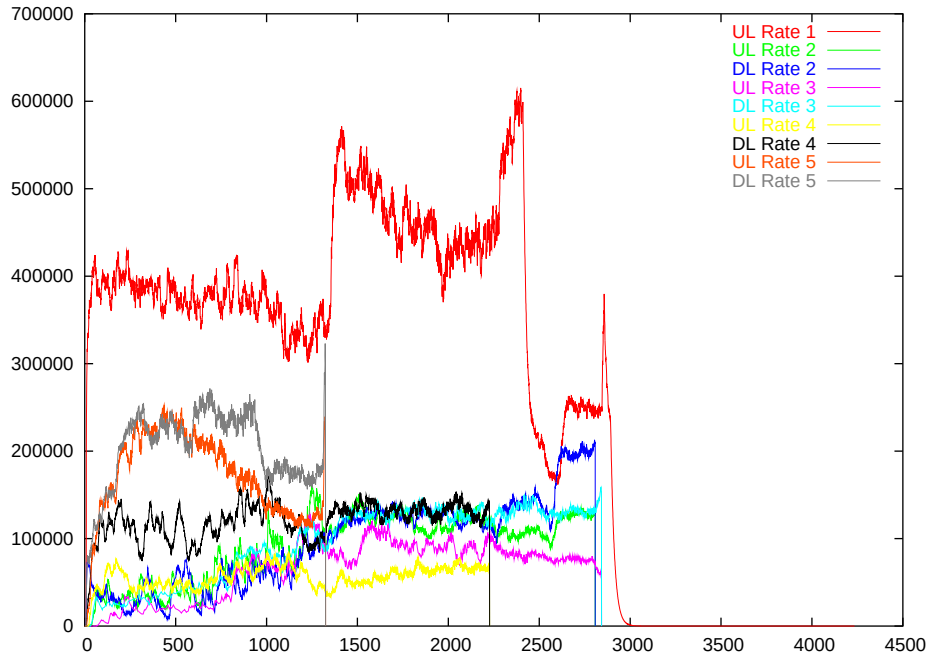

Figure 4. Upload and download Rates of Peers from a seed. In this case, the Seed uploaded almost 4 times the filesize in bytes to 4 peers

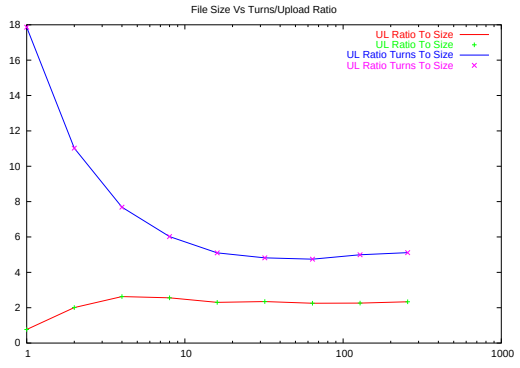

Figure 6. Given files of size 2^0 to 2^8 megabytes in size we see how much the seeder uploads to 4 other peers on a shared network, this is the ratio of turns or uploaded megabytes to the original file size

7 Conclusions

The models generated here are too naive to be really useful when modeling BitTorrent. More complex models would have greater difficulty in proving facts about the system. Although as shown in section **??** even a naive model can successfully model the constraints of the real world. Perhaps a simple model is best as it leaves room for experimental error.

BitTorrent uses concepts from game theory and economics to promote fairness. The use of optimistic strategies enables connections to attempt to renegotiate and counter balance the downward spiraling effects of the tit for tat algorithm.

Optimistic techniques used by BitTorrent seem to be useful strategies to promote file downloading. Bram Cohen, the author of BitTorrent, has suggested that BitTorrent was never meant to promote 1 to 1 upload download ratios, it was created to reduce the load of sharing large files.

In relation to randomized algorithms and analysis of such systems, BitTorrent is quite interesting albeit quite complex. The major parts of BitTorrent related to randomized algorithms are piece picking, upload / download strategy (tit for tat), and hashing. The game theory aspects of BitTorrent relate closely with economics and statistics which are quite related to probabilistic analysis.

8 Future Work

There needs to be further investigation into how a group of peers can collude to create unfair network conditions, such as download more than upload, or even simply disrupt the network enough to disable the distribution of a file. As well there needs to be research into how to detect, prevent and protect against this collusion.

An interesting problem to investigate is whether or not multiple peers on the same computer can download faster than one peer on one computer. Judging by the tit for tat algorithm I have reason to believe that this might be true since there is a tendency to be optimistic and grant a client a bandwidth reprieve to see if they will share more. Thus will this compounded optimism work in the favor of the host with multiple peers versus the host with a single peer? From my personal study of this phenomenon I found that the peers on the same host will share amongst themselves very rapidly.

Given the tit for tat choking algorithms use a 30 second window is there a way to use this timing information to improve the upload download ratio in one's favor?

Often BitTorrent is used across the Internet, over many networks. BitTorrent tests should probably be done across a peer to peer network with explicit paths rather than a shared medium network like 10Mbit ethernet. This should simulate being on different networks and should avoid the problem of limiting 5 peers to a 1 MBit pipe.

9 What Did I actually Do

- Comment and Extract algorithms from BitTorrent. Attempt to understand parts of the program from it's source. Specifically the piecepicker and choker.
- Attempt to read a lot of papers in the area.
- Attempt to run empirical tests on BitTorrent using a network of computer and modified BitTorrent client. Generate a testing framework. Generate the code necessary to collect, retrieve and analyze the data.
- Attempt at proving facts about naive models of BitTorrent. Attempt to verify those models against experimental data.
- Write this report.

References

[1] Pareto efficiency. 2004. http://en.wikipedia.org/wiki/Pareto_efficiency.

[2] A. Adya, W. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment, 2002.

[3] A. L. C. Bazzan and R. H. Bordini. A framework for the simulation of agents with emotions. In *Proceedings of the fifth international conference on Autonomous agents*, pages 292–299. ACM Press, 2001.

[4] B. Cohen. Incentives build robustness in bittorrent. May 2003.

[5] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[6] C. S. Lewis and J. Saia. Scalable byzantine agreement, 2004.

[7] M. Mitzenmacher and E. Upfal. *Probabilistic Analysis and Randomized Algorithms: A First Course*. Brown University, 2003.

[8] L. Mui, M. Mohtashemi, and A. Halberstadt. Notions of reputation in multi-agents systems: a review. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 280–287. ACM Press, 2002.

[9] S. Sen, S. Airiau, and R. Mukherjee. Towards a pareto-optimal solution in general-sum games. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 153–160. ACM Press, 2003.

[10] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. In *Proceedings of the second ACM SIGCOMM Workshop on Internet measurment*, pages 137–150. ACM Press, 2002.

[11] A. P. Toby Donaldson and I.-L. Lin. Emotional pathfinding, 2004.

10 Appendix

Figures 7 , 8 , 9 , 10 , 11 , 12 , 13 , 14 , 15 depict BitTorrent networks working on different file sizes. Do to a bug in the clock, the seeder (the red line) starts earlier but on the graph it appears later. I need to use a synchronized clock in future experiments.

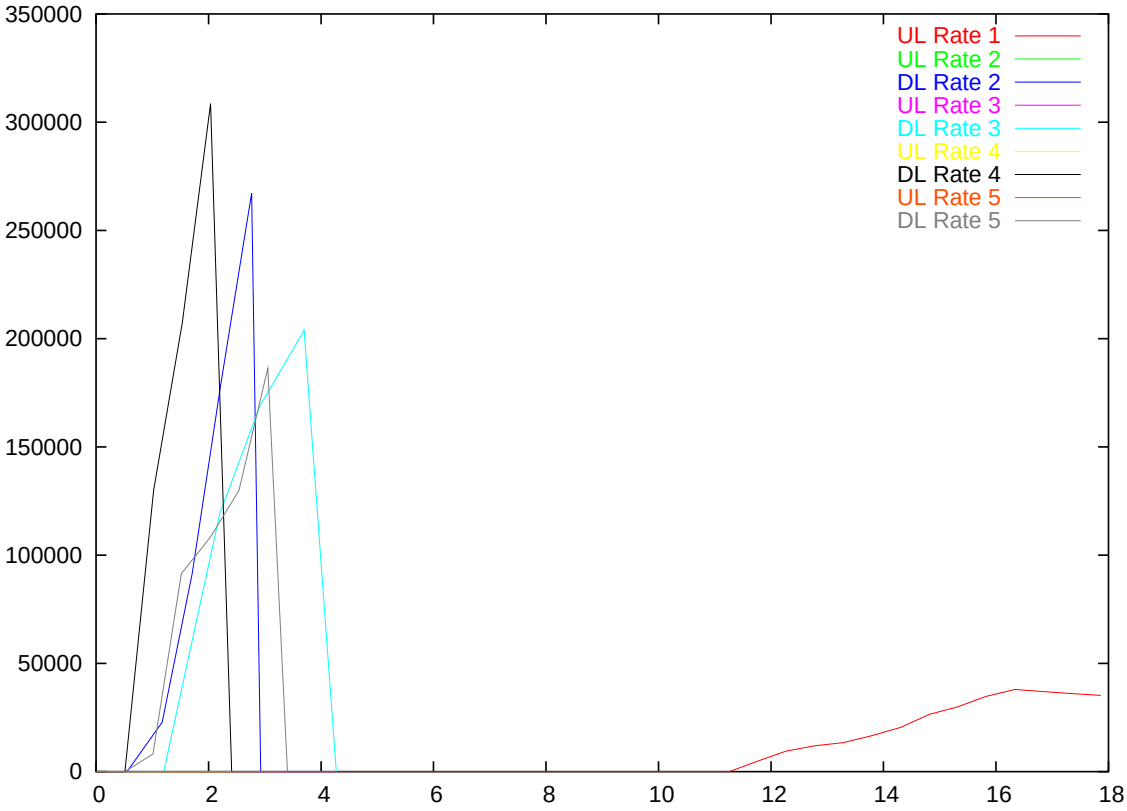

Figure 7. BitTorrent Network of File of 1MB

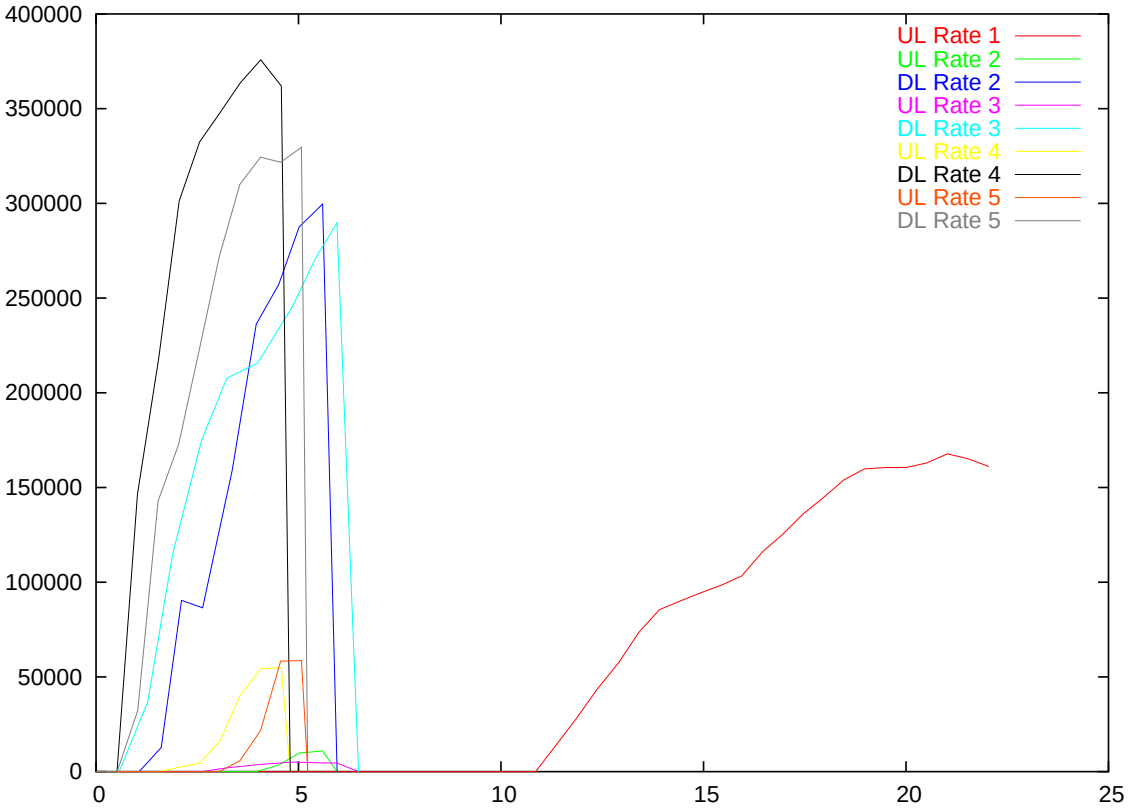

Figure 8. BitTorrent Network of File of 2MB

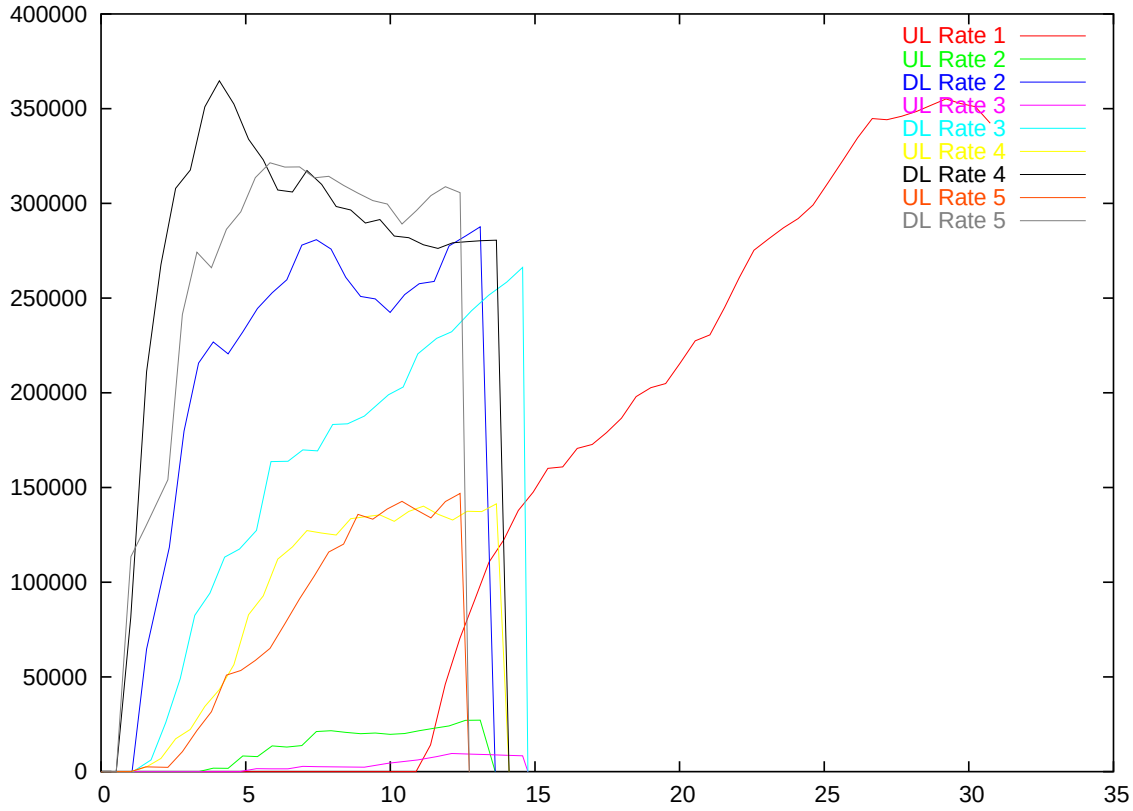

Figure 9. BitTorrent Network of File of 4MB

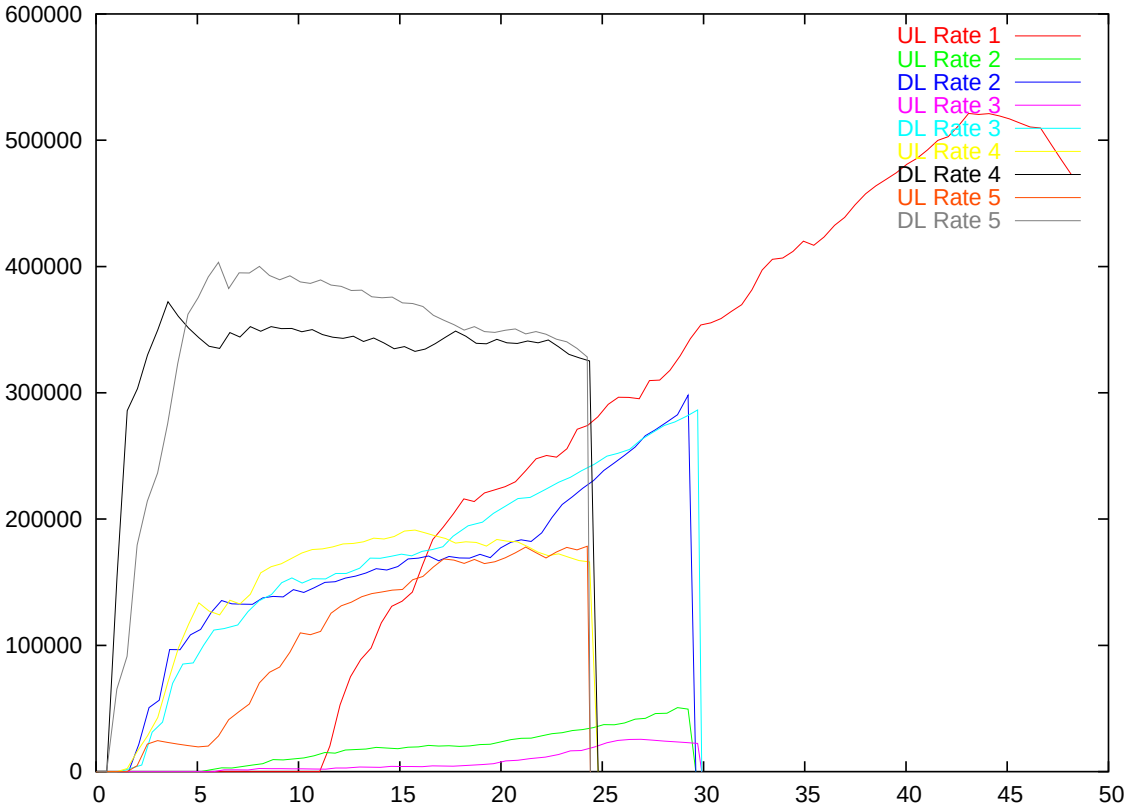

Figure 10. BitTorrent Network of File of 8MB

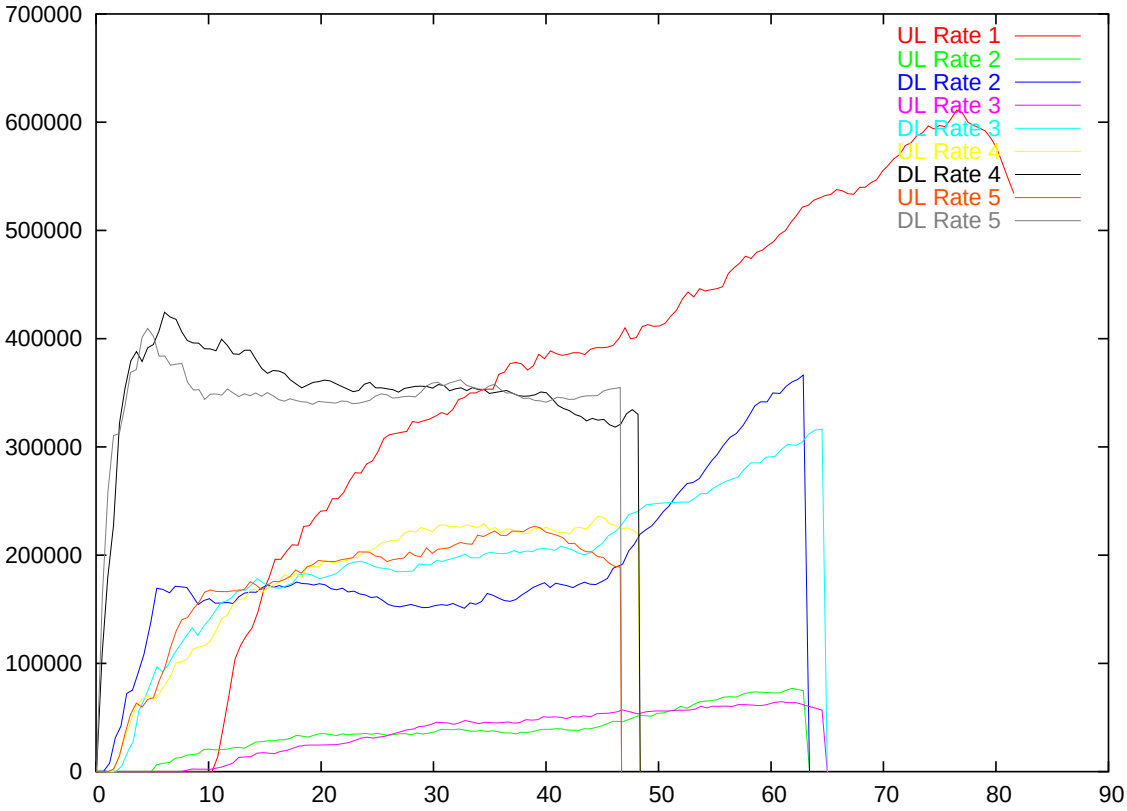

Figure 11. BitTorrent Network of File of 16MB

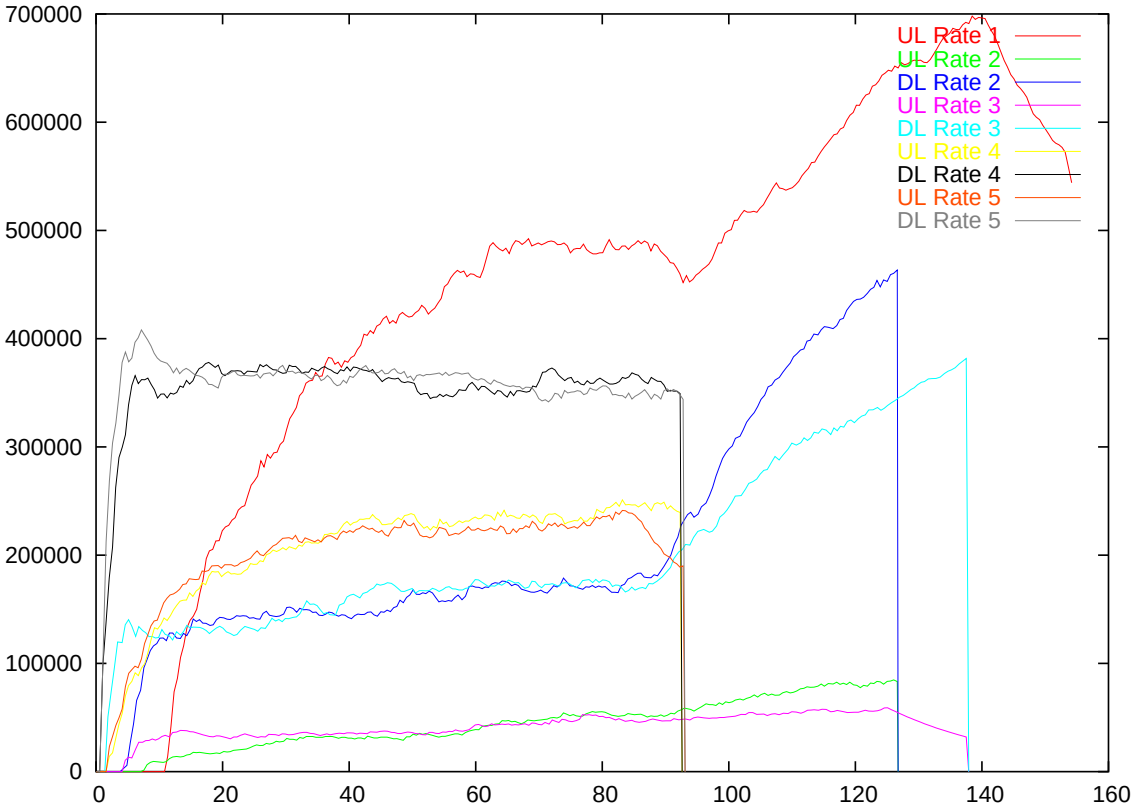

Figure 12. BitTorrent Network of File of 32MB

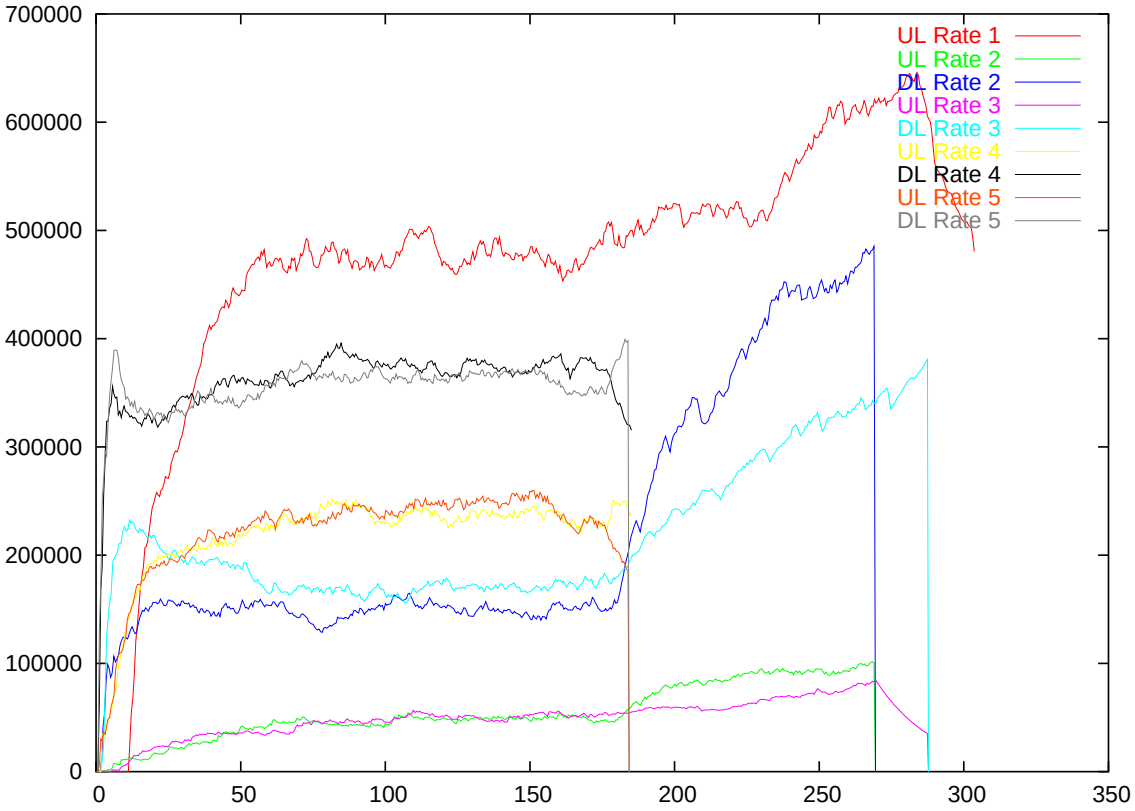

Figure 13. BitTorrent Network of File of 64MB

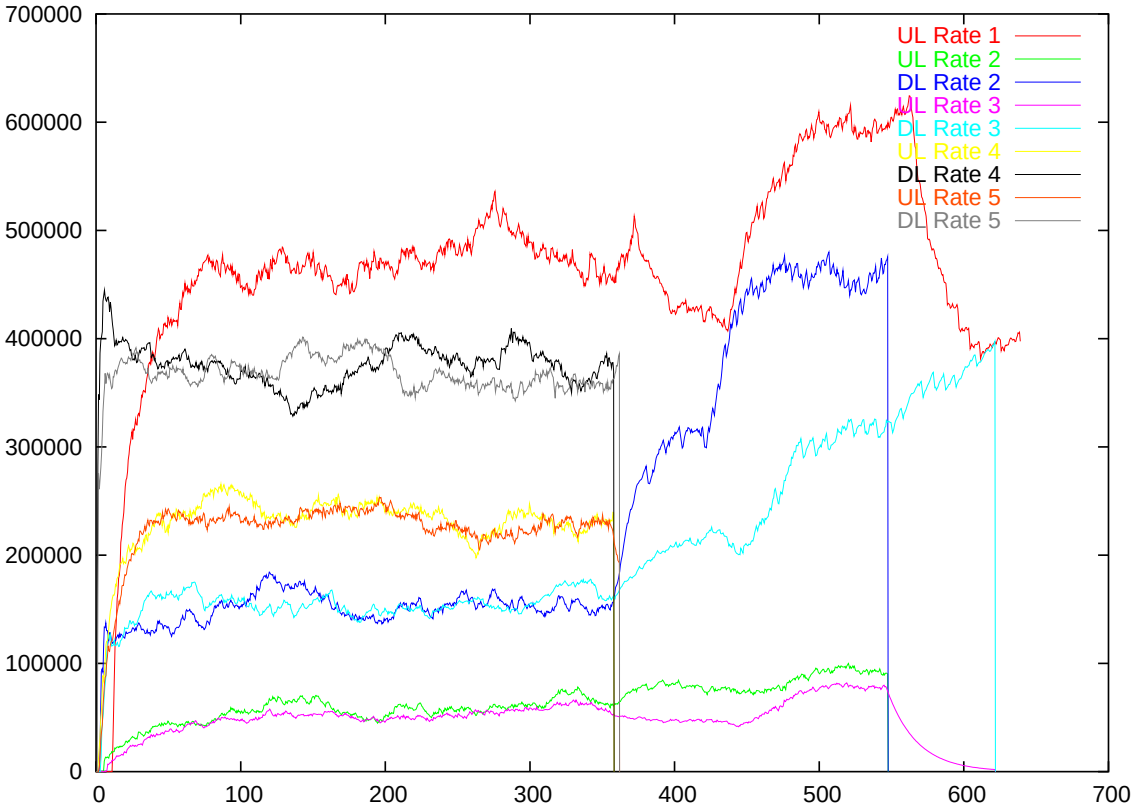

Figure 14. BitTorrent Network of File of 128MB

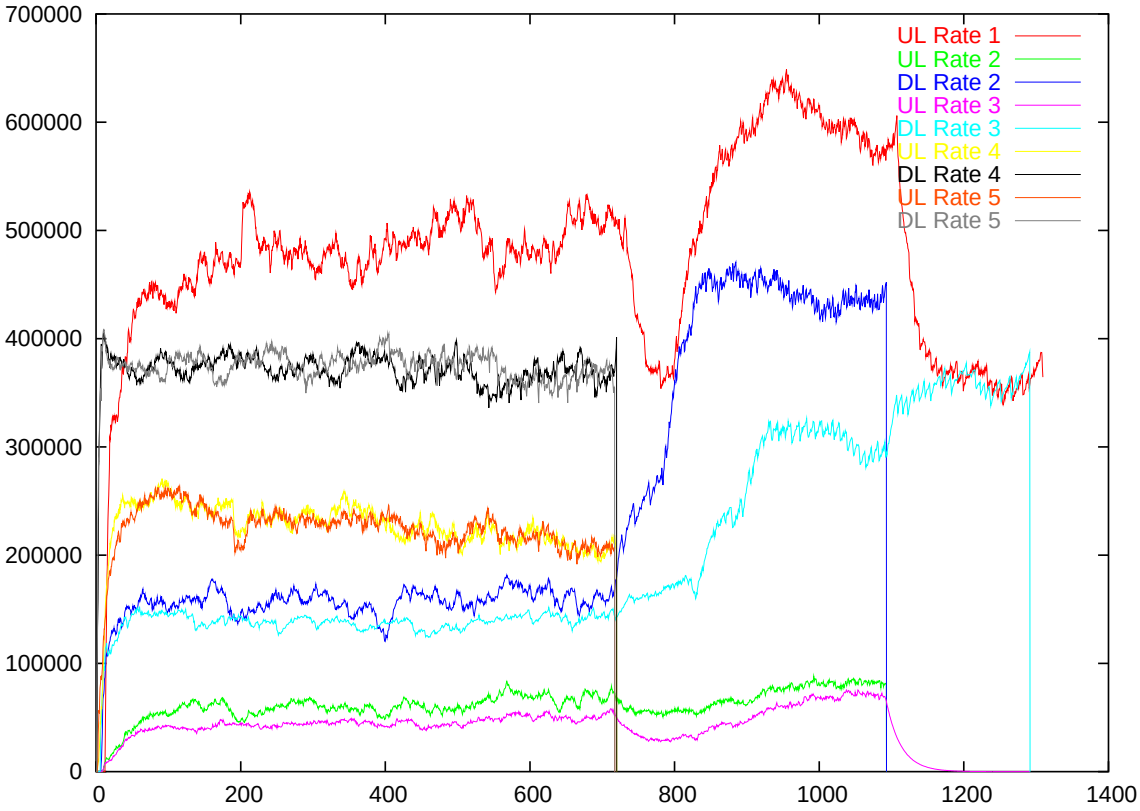

Figure 15. BitTorrent Network of File of 256MB