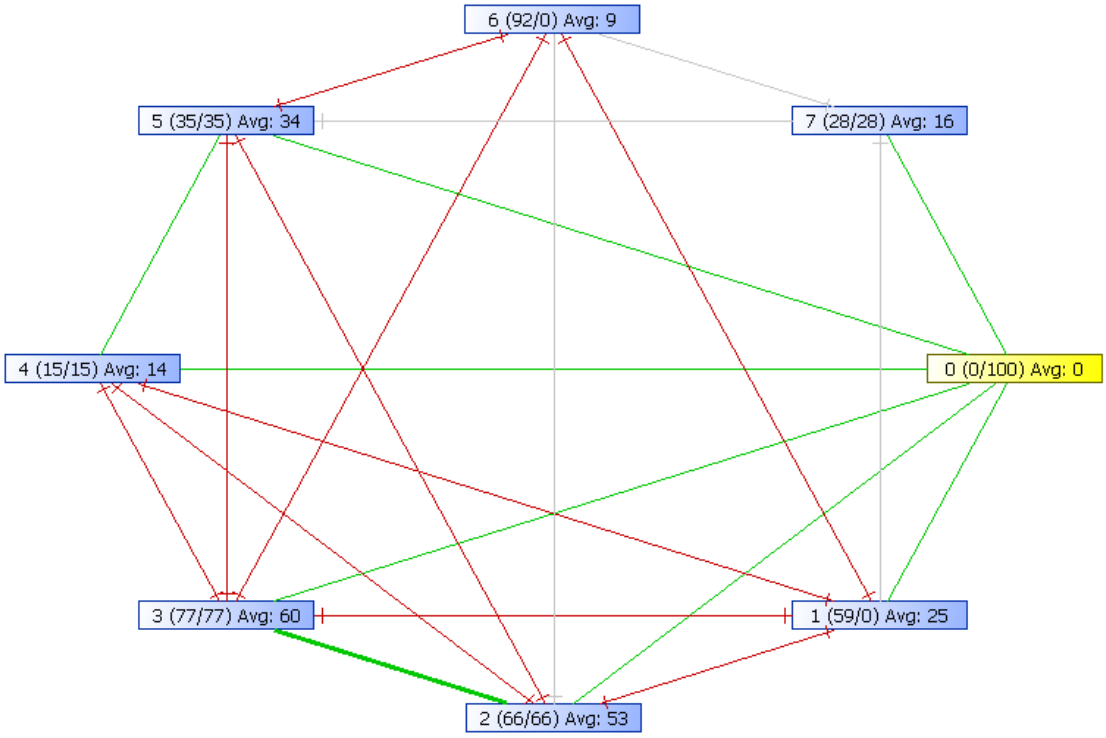
P2P Mechanism Design



Semester Thesis, Lukas Füllemann, SS05
Supervisors: Prof. Dr. Roger Wattenhofer, Stefan Schmid
Department of Computer Science, ETH Zurich

Table of Contents

1 Introduction

The power of peer-to-peer (P2P) systems comes from the fact that many machines are contributing to it. However, if there are no countermeasures, some peers may only profit from the resources provided by other peers while not contributing anything themselves.

Fortunately, such problems are well studied by game theorists. Especially the prisoner's dilemma can be used to show how mutual cooperation leads to the best outcome. The famous game is as follows:

Two suspects A, B are arrested by the police. The police have insufficient evidence for a conviction, and having separated both prisoners, visit each of them and offer the same deal: if one testifies for the prosecution (turns King's Evidence) against the other and the other remains silent, the silent accomplice receives the full 10-year sentence and the betrayer goes free. If both stay silent, the police can only give both prisoners 6 months for a minor charge. If both betray each other, they receive a 2-year sentence each [1].

This game illustrates the fact that it is better to cooperate than being selfish; but only if the opponent cooperates as well. The question is how to build incentives or punishments to make selfish people contributing to the system, e. g. providing disk space, CPU cycles or network bandwidth. This art of designing rules to achieve a specific outcome is also called mechanism design. This aspect is important, because free riders are not willing to contribute towards the cost of public goods when they hope that someone else will bear the cost instead. This leads to degradation of the performance of the system.

I focus in this thesis specially on file sharing and file distribution systems. Generally such systems are used by few millions of users not knowing each other and normally having no reason to contribute anything. After looking at general problems in Section 2, I analyze existing file sharing systems in Section 3 and study how the problem of free riders is solved. In Section 4 I present an analysis and improvement of BitTorrent, which I compare with a personal proposition by simulation in Section 5. The ability of creating a fast and completely fair system will be one oft the most surprising results.

2 General Problems

Apart from mechanism design, there are some general constraints assisting the free rider problem:

2.1 Asymmetric Network Connections

Most of the users today use an asymmetric network link to connect to the internet. The download rate currently is four to ten times higher than upload rate.

A transferred file needs as much upload bandwidth at one node as download bandwidth at another node. Globally seen, the output of the system equals the input of the system. Therefore the upload bandwidth becomes the bottleneck and the download operates far away from the expected full capacity. This means you normally can not utilize your download speed because you are not able to contribute enough.

2.2 NAT-routers / Firewalls

NAT stands for Network Address Translation and is a standard to separate local networks from the internet. With this technique it is possible to connect many local computers with a local IP to the internet with just one IP address. Although having many practical and security relevant advantages, peers connecting from outside are blocked with factory settings. Peers behind such firewalls involuntary become free riders in systems not taking care of this problem or without special configuration of the router, even if they would like to contribute.

Some systems wisely care about this problem, other leave it just to the user to solve it. Gnutella, for example, addresses it by introducing a push download: If peer A wants to download a file from peer B being behind a firewall or a NAT-router, A can send a push message to B, which connects out directly to A and performs the upload.

2.3 Copyright Infringement

The current law in Switzerland prohibits uploading copyright protected content to other users because this is treated as a publication and is therefore illegal. Sharing is only allowed between friends and relatives [2]. On the other hand, downloading is grey area and no conviction happened yet. Accusations from the music industry were always pointed to user releasing files and never to those only downloading.

Because of this reason many people are not willing to provide files, but dare to download from the system. It is not always easy to distinguish between download and upload as in most of the systems it is not possible to download files without providing them to other users – at least not until the download has finished.

3 Analysis of Current and Past Systems

3.1 Systems without Restrictions

3.1.1 Napster

Napster as one of the oldest p2p file sharing system (Mai 1999, [3]) does not contain any mechanisms restricting free riders. It uses a central index for searching, but downloading happens peer-to-peer. Because of this central index it was possible for the music industry to filter out copyright protected content and the users switched to other systems.

3.1.2 Gnutella

Unlike Napster, Gnutella is a pure P2P-Network. Both searching and downloading happens peer to peer without a central server. This distributed search lead in the beginnings to problems of stability and search relevance but was an important step to get out of control of the music industry. It uses a very simple protocol based on five message types: 'Ping'/'Pong' messages are used to actively discover hosts on the network, 'Query'/'Query-Hit' messages for searching and 'Push' messages for traversing firewalls by initiating downloads from the host providing the specific file.

Gnutella neither contains any mechanisms controlling free riders. This system is well studied and measurements [4] have shown the reality of the free-rider problem: 70% of the users do not share any files and nearly 50% of all responses are returned by the top 1% of sharing hosts.

Gnutella is still heavily in use by about 2 million users (Dec. 2005) and ranks among the top three most used networks behind eDonkey2k and Fasttrack [5].

Many propositions (such as nodes controlling their neighbours) were made to improve the behaviour against free riders but none of them were ever implemented in the productive system.

3.1.3 Freenet

Freenet is a decentralized censorship-resistant peer-to-peer distributed data store. It works by pooling the contributed bandwidth and storage space of member computers to allow users to anonymously publish or retrieve various kinds of information [1].

Users contribute to the network by giving bandwidth and a portion of their hard drive for storing files. Unlike other p2p file sharing networks, Freenet does not let the user control what is stored in the data store. Instead, files are kept or deleted depending on how popular they are, with the least popular being discarded to make way for newer or more popular content. Files in the data store are encrypted to reduce the likelihood of prosecution by persons wishing to censor Freenet content.

The amount of disk space providing to the network can easily be changed in the configuration. I was unable to state clearly what the incentive is to provide disk space except from having some queries replied from the local disk.

The proposed action [6] to get better performance is to set up a node as permanent. The node will connect itself better to the rest of the network, so requesting keys will be faster. Since you will be getting a lot of traffic it is likely that the newly stored material will be closer to you (or even on your very own node) and you will have faster access to the data stored in the network.

To sum up, Freenet does not exclude free riders but offers then only poor performance.

3.2 Credit based Systems

3.2.1 Mojo Nation

Mojo Nation ([7], [8]) was one of the first systems that used a more sophisticated mechanism design by introducing artificial money called “Mojo”. Mojo can be earned by offering one or more of the following resources to the network:

- Block-Server: Provides disk storage for file blocks
- Content Tracker: Tracks the location of content
- Publish Agent: Service for publishing content into the Mojo-Nation-network
- Relay Server: Forwards messages (for increased security)

This earned money can now be used for downloading or publishing files. Yes, publishing is not for free, because many of the services listed above are involved!

This system requires cleverly devised encryption mechanisms to ensure the accuracy of all Mojo balances and transactions involved.

In a second step it was planned to bridge the gap to real money. This opens up many new possibilities. For instance it would be possible to earn money by simply running a computer connected to the internet offering one ore more of the services listed above.

Additionally to the micro payment system it was intended to introduce a reputation based mechanism where users can rate other users like in eBay or Ricardo. This should prevent users to offer scrap and useless data.

This systems never got into a running phase and leaves many questions open. Who decides which file chunk is stored where? Who sets the fees for the services? How can the payment system be used to accelerate the system by using the fastest links? Is this system stable or are the fares generally decreasing until no one is willing to contribute? Does the traditional idea of bid and demand hold in this system?

The successor of MojoNation, MNet does not use the payment system any more, it is announced in a second step.

3.2.2 eDonkey / eMule

The protocol definition of eDonkey was kept secret and explored by reverse engineering. This is the reason I analyze the open source and well documented eMule-protocol which is based on the eDonkey-protocol.

eMule [9] uses a credit system to encourage users to share files. When peer A uploads a file to peer B, B updates locally the credit of peer A according to size of the file downloaded. This credit system is not global - the credit for the mentioned transfer is kept locally by client B and will be taken into account only when the uploading client A (which earned the credit) is going to download from client B. This means that peer A and B pair wise store the credit of each other. Credit is calculated as the *minimum* of:

$$ratio1 = \frac{2 \cdot uploaded\_total}{downloaded\_total}$$

When downloaded total is zero this expression evaluates to 10

$$ratio2 = \sqrt{uploaded\_total + 2}$$

When uploaded total is less then one MB this expression evaluates to 1

The upload/download amounts are calculated in megabytes. In any case the credit is in the range [1..10].

The earned credits do not influence download speeds but they influence the position in the other peer's waiting queue. Users with high credit generally do not have to wait for long while free rides (characterized by low credit) get queued and may be passed by other users.

One pretended method to free ride is to keep the shared folders always empty. This means you just move a file after successful downloading to a private folder. This is a probable method if you are looking for unpopular files. Otherwise you get queued and probably have to wait for a long time while being passed by other peers with higher credits.

3.3 Participation level based Systems

3.3.1 FastTrack / Kazaa

FastTrack and its first client Kazaa ([1]) were introduced in 2001, just before Napster shut down. It was the most popular protocol in 2003. It uses encryption and was not documented by its creators. Because sensitive initialization data for the encryption

algorithms is sent in clear text, reverse engineering was relatively easy and in 2003, the client-supernode (=client with server jobs) communication was completely explored.

Kazaa uses roughly the same mechanism design idea as the eDonkey-client eMule: It looks at the amount up- and downloaded, but handles it in a very different way. While eMule calls it 'credit' and it is pair wise stored at the other client, Kazaa uses a global score called 'participation level' [10]. Interestingly, both values are calculated similarly:

$$participation\_level = \frac{upload}{download} \cdot 100\% \qquad \text{Range: 1..1000}$$

This level must be understood as a representation of the user's recent sharing pattern of the last seven days and is not accumulated over long periods of time. Upgrading to a newer version of Kazaa resets the level to 'medium'.

Officially, the participation level is only used to set the search radius for files in the network, other sources talk about queue reordering and download priorities.

Participation Level	**Number of 'Search Mores'**	
	Kazaa	Kazaa Plus
Low (0-50)	1 x	2 x
Medium (51-100)	2 x	4 x
High (101-200)	3 x	6 x
Guru (201-500)	4 x	9 x
Deity (501-900)	5 x	12 x
Supreme Being (901-1000)	5 x	15 x

Kazaa is the free version, Kazaa Plus the premium version available for about 30$.

Unfortunately, this system has one big problem: the participation level is stored at the user's machine. This opens up all doors for cheating. And those doors were used. Several hacks exist on the web for the original client. Another possibility is to use K++ (former Kazaa Lite), an unauthorized modification of the Kazaa Media Desktop application which excludes adware and spyware. It includes in later versions a memory patcher that removed search limit restrictions, multisource limits, and set one's participation level to the maximum of 1000.

Keeping in mind that it is now (fall 2005) as almost as widely used as the official Kazaa client itself, the world full of Supreme Beings (see table above) is guaranteed. Another possibility of cheating would be reinstalling the application and get again a 'medium' level if the level has fallen down to 'low'.

3.4 Advanced Systems

3.4.1 BitTorrent

BitTorrent was designed as a file distribution system (e.g. for Linux ISOs) and not primarily for file sharing. Its goal is to minimize traffic at the server side and to distribute it to clients. For that reason the data is chopped into chunks and once a peer

has downloaded a chunk from the initial source, it offers it for downloading to other clients while getting chunks from other peers.

The main goal of BitTorrent concerning efficiency is to be Pareto efficient, because this measure is used often by economists as efficiency goal. By definition an outcome of a game is Pareto efficient if there is no other outcome that makes every player at least as well off and at least one player strictly better off. That is, a Pareto Optimal outcome cannot be improved upon without hurting at least one [11].

In computer science terms, seeking Pareto efficiency is a local optimization problem 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 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.

To understand BitTorrent, we go back to the prisoner's dilemma from the introduction. In the iterated version of this game, the suspect may react in the current round to the other's decision of the last round. If he initially cooperates and then always decides the same as the opponent's previous action, its strategy is called tit-for-tat. If the opponent previously was cooperative, the agent is cooperative as well in this round. If the opponent was not, the agent is not cooperative too. Tit-for-tat turned out to be the most effective strategy in various simulations created by teams of computer scientists, economists, and psychologists. Whether this strategy is optimal or not has not been proved yet [1].

One peer in the BitTorrent-Network can now be seen as one agent playing pair wise with other peers an iterated prisoner's dilemma. Cooperation amongst peers means uploading, and non-cooperation is where "choking" (temporary refusal to upload) comes in. This makes sense as the goal is to have several bidirectional connections running continuously by uploading to peers, which have uploaded to this peer recently. Unutilized connections are uploaded to on a trial basis to see if better transfer rates could be found using them.

Choking Algorithms

As a basic mechanism, BitTorrent always unchokes a fixed number of connections (default is four), the rest of the connections will not be terminated but are only used for downloading. Unutilized connections are also uploaded to on a trial basis to see if better transfer rates could be found using them. This decision which peers to choke is based on a 20-second average of the download rates and may change every ten seconds. This amount of time is needed for the connection to get to full speed and leads to more stability. This unchoking algorithm pretends to ensure the tit-for-tat semantic and highly removes free riders as peers not uploading will be choked the most. As we will see later, this design does not have all properties tit-for-tat comes with.

The other rule of the game, the cooperation in the first move, is also recognizable in the algorithm and is called 'Optimistic Unchoking'. Every peer maintains a single optimistic unchoked peer, which is unchoked regardless of the current download rate from it. This peer is rotated every third rechoke period (30 seconds). In this period the upload has a chance to get to full capacity, the other peer to react and the download from that peer to get to full capacity.

The third part of the algorithm tells a peer what to do, if he occasionally gets choked by many or all peers. If he didn't get any data from a particular peer, he doesn't upload to

it any more except as an optimistic unchoke. The peer is removed from the unchoke candidates and the connection is used for an additional optimistic unchoke. This can be seen as a fast recovery [12].

Although those rules look quite simple, the algorithm bases on one hand on well thought-out concepts of game theory and on the other on small technical details like timings or connection management.

One weak point of the system could be the tracker, which passes addresses of existing peers to new peers of the system. Without restriction on the number of peers returned, it could be possible for a peer to get a lot links and therefore benefiting just from optimistic unchoking without contributing anything.

3.5 Comments on Described Systems

Looking at those systems, we see that in the beginning (e.g. Napster and Gnutella) the notion of mechanism design does not show up. A high free rider ratio was the outcome and people started thinking about controlling mechanisms.

Some system designers like those of Mojo Nation invented sophisticated but complex money systems leading to high encryption and transaction overhead. I'm not surprised by the fact that this system never got into a running phase, as managing safely money accounts in a reliable peer to peer system combined with a fair fare balancing is too complex for just sharing files.

Kazaa considers the amount of data up- and downloaded and introduces the notion of participation level. The idea behind this concept is good, as not every link has to be 'fair', which means that a peer may get a lot from one peer without uploading to it if it uploads concurrently to another peer or has uploaded much earlier. This system decouples up- and download in time and space. This is a great idea, but storing this level is only possible at the user's machine itself, which opens up all doors for cheating. Therefore this solution unfortunately is not useful in practice. This shows the fact, that a peer always has to be controlled by it's neighbouring peers and possible persistent information has to be stored as well at the direct connected peers.

eMule started wisely to look at single connections and tries to establish fairness in the small. Its credit system does not control bandwidth directly but creates incentives by changing the peer's positions in waiting queues.

BitTorrent finally goes one step further and uses download speed as incentive for uploading. Because its algorithm is based on game theory, BitTorrent seems to be the 'best' current system regarding fairness.

4 BitTorrent Revised

4.1 BitTorrent and tit-for-tat

Axelrod [13] analyzed the best algorithms for the iterated prisoner's dilemma and stated the four following properties:

- They are *nice*, which means that they cooperate as long as the opponents do. In other words, they never defect first.
- They are *retaliatory*, which means that they stop cooperating if the opponents defect. This property prevents them from being exploited

- They are *forgiving*, which means that they cooperate again, if the opponents resume cooperating after mutual defection. This property helps reconstruct trust.
- Their behaviour is *clear*, which means that they signal to the opponents that they act reciprocally. This property leads the opponents, particularly those who "probe" others, into cooperation.

Comparing these properties with the algorithm described above, we see that the algorithm does not meet all of them.

- With the optimistic unchoking BitTorrent may found a better link than the currently unchoked and replaces the worst of them. This hurts the *nice*-property as our peer defects first. This problem is mitigated with the optimistic unchoking as the previously choked peer may be selected again for optimistic unchoking and gets the chance to get unchoked again, if the other rates decreased.
- On the other hand, the optimistic unchoking keeps the mechanism from being retaliatory. If it is not compensated for, a 30-second unconditional upload can be a significant resource leak. As the optimistic unchoked peer is chosen in a round-robin fashion, free riders are given the chance to benefit repeatedly.

4.1.1 A theoretical bound

This possibility of benefiting from optimistic unchokes can be expressed mathematically [14]: We consider a network with N peers that have the same uploading bandwidth r. We assume that each peer has k uploads and one optimistic unchoking upload. Now, a new peer f with zero uploading bandwidth joins the network. Each peer will randomly choose a peer that it is not currently uploading to as the target of its optimistic unchoking. So, for each peer, on average $\frac{1}{N-k}$ of time it will upload to f. Summing up over all peers, the download rate of f will be

$$N \frac{1}{N-k} \frac{r}{k+1} \approx \frac{r}{k+1}$$

when N is large. In current BitTorrent, $k = 4$ and thus a free rider gets 20% of the possible maximum downloading rate.

One possibility to reduce this rate is to increase k. This means that a lot of concurrent TCP connections have to share the same bandwidth and thus more time-outs and poor performance may occur.

Another possibility is to enhance the torrent by limiting the number of returned peers when a peer asks for more peers to connect to. The effect on performance is subject for further study.

4.2 Deficit Mechanism

Based on the fact that BitTorrent does not fulfil the four properties 'nice', 'retaliatory', 'forgiving' and 'clear', Jun and Ahamad proposed a slightly different algorithm [15] we call deficit mechanism or just DEFICIT.

It works as follows: Each peer maintains the upload amount u and the download amount d for each link. The *deficit* of a link is defined as u - d and is always restricted up to a certain bound:

$$u - d \leq f \cdot c$$

where *f* is called *nice factor* and *c* denotes the size of a fragment. Within this condition, the peer uploads evenly to all links as much as it can. The nice factor determines the amount that a peer is willing to risk for a chance to establish cooperation. Although neighbours may be tempted to take advantage of this 'nice' peer, they will benefit more through the repeated exchange of fragments if they cooperate.

This mechanism looks much simpler and clearer than BitTorrent. Looking again at the four properties, we see that this mechanism meets all of them.

5 Simulation

5.1 Motivation

Based on the theoretical bandwidth free riders can get in BitTorrent and by the proposed deficit algorithm, I want to analyze which algorithm behaves better regarding performance and fairness. Because real life measurements or a simulation e.g. on PlanetLab [16] would exceed the work of this thesis, I wrote a multithreaded Java program to simulate different algorithms.

5.2 Experimental Setup

The simulation can be either run with a graphical user interface to see the algorithm working or run in measurement mode with hundreds of client to get more precise results.

For sake of simplicity I made a few abstractions:

- The graph representing the peers and connections is static, which means no peers join or leave the simulation during the execution phase. Additional the bandwidths of the participating nodes are constant.
- The simulation does not care about fragments of the file; they are treated as arbitrarily small. Furthermore it is not important, from whom a peer downloads; we assume all chunks are available everywhere. This is why I can hide the chunk selection algorithm.
- As the word simulation suggests, no real TCP connection are made. To get a reasonable behaviour, the downloading node in a connection tries to maximize the speed by asking for more bandwidth (if spare bandwidth is left), while the uploading client tries to distribute it's bandwidth as evenly as possible to all connected nodes without overwhelming them.

5.3 Algorithms

As indicated in the motivation, I want to compare different algorithms. They are:

ALTRUISTIC — This is the reference algorithm. A peer distributes its upload bandwidth evenly to all links. There is no restriction against free rider in this system.

BITTORRENT — This is a simplified version of the BitTorrent protocol. A peer opens up 3 connections to its best peers (those he get the highest download rates from) and maintains one optimistic unchoke as described in the BitTorrent section.

DEFICIT This is the proposed algorithm of Jun and Ahamad. Each peer maintains the deficit for every link and provides not more data than received by this link plus nice factor (here: nice factor = 1)

RATE This is my personal proposition. It bases on the DEFICIT algorithm but prevents the connections from closing and opening up again and again by adapting the speed and making a links symmetric. A peer provides at most one KB/s more than downloading from a single link. The motivation of this algorithm is to get a better performance than DEFICIT by smoothing the speed.

5.4 Simulation Environment

5.4.1 GUI mode

This mode is just for understanding the tested algorithm and for seeing how they work. The network consists of one tracker (not visible), one seed and a few other nodes running the same algorithm

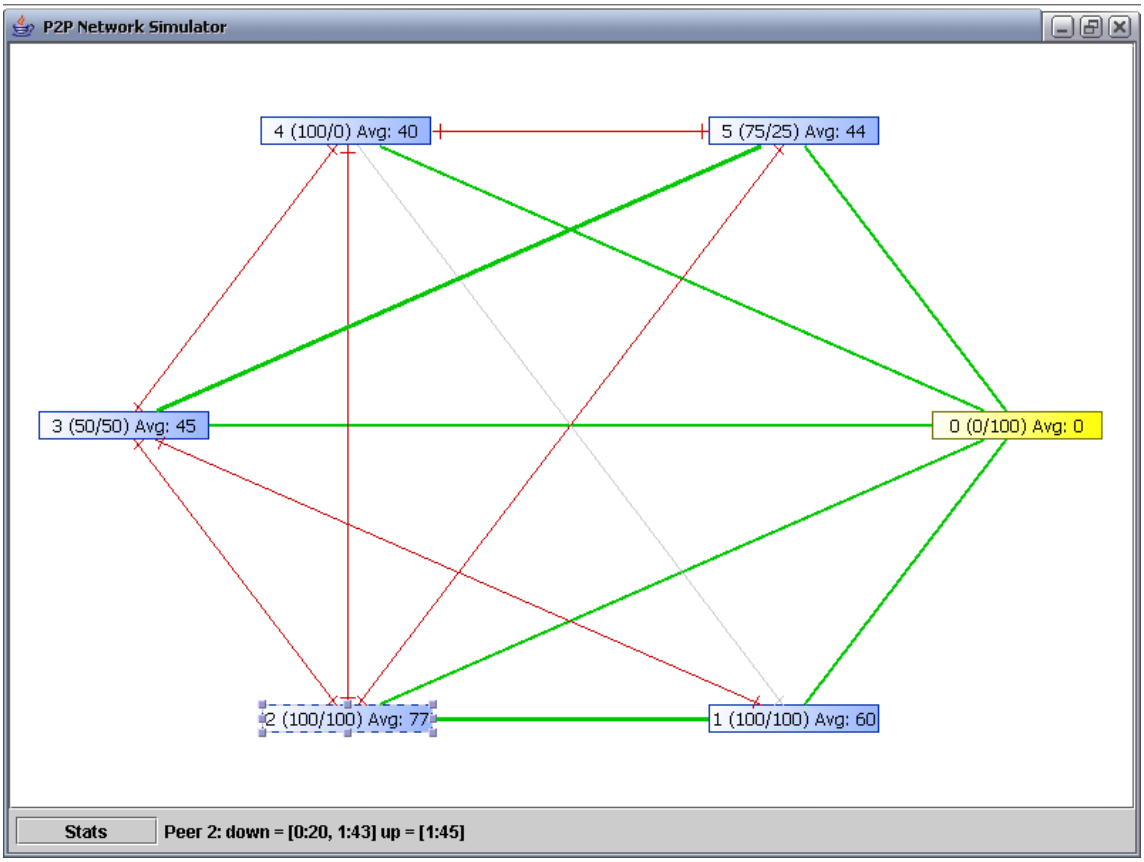


Figure 1: GUI mode

Each vertex represents a peer and each edge a bidirectional connection. Colours of the vertices indicate the algorithm running on that vertex. Above we see one seed (coloured yellow) and five vertices running BITTORRENT (coloured blue). The label of a vertex consist first of the index of the node followed by the link speed (download/upload) and finally the average download rate. The peers in the example above have the following roles:

- Peer 1 and Peer 2: nodes with fast symmetric connections
- Peer 3: Slower peer, but yet symmetric
- Peer 4: fast peer, but only able to download (free rider)
- Peer 5: Slower peer with asymmetric link (faster download than upload)

The small bar at the end of an edge indicates that the peer located at the bar keeps the connection closed. In Figure 1, peer 4 for example is unable to download from peer 5, because the connection is closed by peer 5 and vice versa.

The colour of the edges (if visible) is coupled to the bars. Green means connection is open for both peers, red indicates a connection closed by both sides and gray a one-sided closed connection

The width of the edge finally shows the current rate of the bigger link. Remember an edge consists of two different unidirectional links. Thresholds are 20KB/s for the medium respectively 40KB/s for the thickest width.

As the network is very small, the number of unchoked connections in BITTORRENT is reduced to two.

5.4.2 Measurement Mode

With this mode I compared the different algorithms. I do not use a seed, as it tampers the results and always offers a constant rate to each peer. I always created 100 peers and the tracker always returns four of the already created peers which leads to seven neighbours on average. The download rates are chosen uniform at random from the interval [0...100]. The upload rate is either equal to the download rate or zero in case of a free rider. The simulation was executed for free rider ratios from zero to 90 with step size ten.

5.5 Metrics

To be able to compare the four described algorithms I define two masses of measurement:

Fairness

A good algorithm has to be fair. This means a peer should not be allowed to download more than he uploads to the network. Hence the more he downloads than he uploads the bigger the penalty gets. Making the penalty bandwidth independent and subtracting from one leads to the fairness of a single peer:

$$fairness(peer\,i) = \begin{cases} 100\% & \text{if } avgDownRate < upRate \\ (1 - \frac{avgDownRate - upRate}{downRate}) \cdot 100\% & \text{else} \end{cases}$$

Forcing the fairness for each peer to be smaller or equal 100%, we ensure no penalties may be cancelled out. Calculating the average of the fairness of all peers leads to the fairness of the system:

$$fairness_{system} = \frac{1}{N} \sum_{\forall\, peers} fairness(peer\,i)$$

Performance

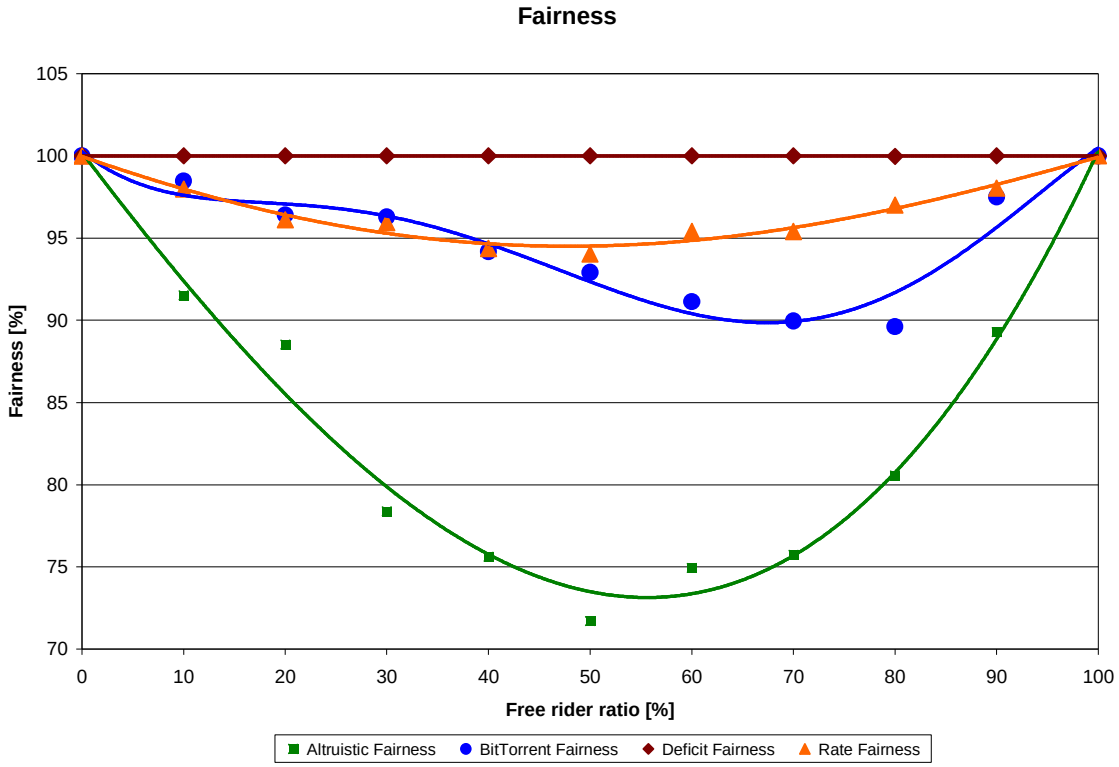
A good algorithm has not only to produce high fairness in the system; the system also has to be reasonably fast. Ensuring fairness should not slow down the system too much. In a first step, we just use the average download rate of the system:

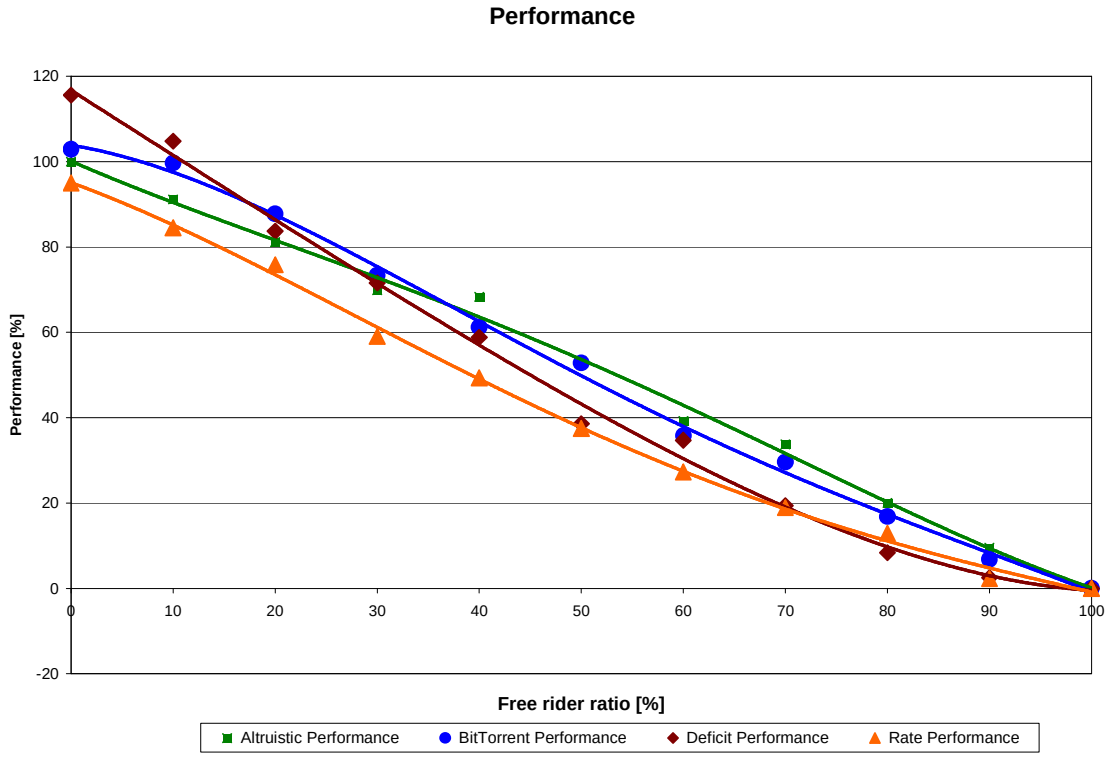
$$avgDownloadRate_{system}(algorithm\ a) = \frac{1}{N} \sum_{\forall\ peers} avgDownloadRate(peer\ i)$$

To compare the different algorithms, I standardize the average download rates of the different algorithms by dividing by the average download rate of ALTRUISTIC. 100% has now the meaning of the performance of a system without any restrictions:

$$performance(algorithm\ a) = \frac{avgDownloadRate(algorithm\ a)}{avgDownloadRate(altruistic)}$$

5.6 Outcome of the Simulation





5.7 Analysis

5.7.1 General

Looking at the fairness graph, we see huge differences between the algorithms. DEFICT really holds its promises and offers a fairness of 100% for all free rider ratios while the fairness of ALTRUISTIC goes down to about 70%, because no countermeasures against free riders are used. The other algorithms are quite good and do not go much below the 95% line. It probably will be the performance going to decide the battle among them. And it indeed does: BITTORRENT is always about 10% faster than RATE. The expectation of ALTRUISTIC being the fastest one does not hold at all, BITTORRENT and DEFICIT perform much better up to a free rider ratio of 40%.

The fairness gets better again with high free rider ratio, because only few nodes upload data and therefore average download rates are small compared to the possible downloads rates causing a high fairness (see Section 5.5).

5.7.2 ALTRUISTIC

This algorithm is only used as a reference to see how a system without mechanism design behaves. Therefore the fairness is the worst of all but stays for all free rider ratios above 70%. This means in general a peer is able to download 30% more than he uploads. Concerning performance, it starts moderate and gets the best with increasing free rider ratio.

The absolute average download rate (not visible in the chart) for the case with no free riders is 41. As the download rate distribution is uniform from the range [0..100], its expectancy value is 50, so altruistic gets about 82% of the possible download speed. This comes from the non-completeness of the graph a non-optimal resource allocation.

5.7.3 BITTORRENT

The fairness is much better than the theoretical bounds predicted. Up to the free rider ratio of 40% it stays above 95%. Higher ratios are not likely to happen in practice and are just for visualizing the whole spectrum. This means a user in a BITTORRENT network is able to download 5% more than he uploads. Also regarding performance it is always under the top two, in the beginning even higher than altruistic.

5.7.4 DEFICIT

This algorithm meets the target of being fairer than BITTORRENT. It is almost perfect and independent of the amount of free riders. A peer is indeed only able to download the amount he uploads. The nice factor vanishes for a longer execution time.

This algorithm seems to suit best for systems with few free riders. It is about 15% faster than ALTRUISTIC in the case of no free riders, which is an extremely good result.

5.7.5 RATE

Even though this algorithm is just a small modification of DEFICIT, it is much worse concerning fairness and performance. Because it offers always 1KB/s more to its links than receiving from it, a free rider is able to get a constant rate from every peer. That's why the fairness is not as good as in DEFICIT, but it does not go below 95% and is still feasible. The performance is about 20% lower than in deficit with no free riders and closes up the gap at a free rider ratio of about 70%. The reason why RATE is much slower than DEFICIT is shown in the next section.

5.8 Speed Discrepancies

We saw in the analysis above that the performances of the different algorithms behave unnatural. BITTORRENT and DEFICIT are faster than ALTRUISTIC and RATE. We expected the ALTRUISTIC to be the fastest and RATE being faster than DEFICIT because of the avoidance of the connection resetting.

Looking closer at the algorithm description in Section 5.3, we see that they have a different connection management. BITTORRENT and DEFICIT have the opportunity to set the upload rate to a peer completely to zero. This indirectly gives other peers the chance to increase their rate temporarily higher than they would get if the peer distributes his rate evenly to all links.

I want to illustrate this fact with a small toy network. Each node has a symmetric connection with rate 3. The question is now how to distribute this rate to the available links. The numbers on the edges show the symmetric utilisation of that edge. This can basically be done in two manners:

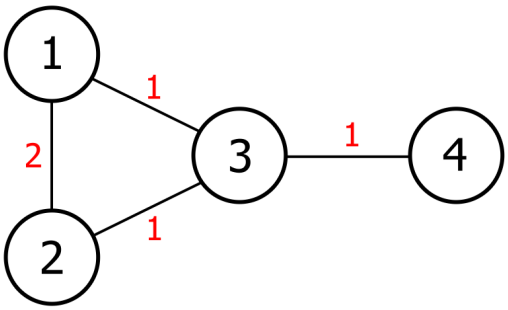


Figure 2: Even distribution of bandwidth

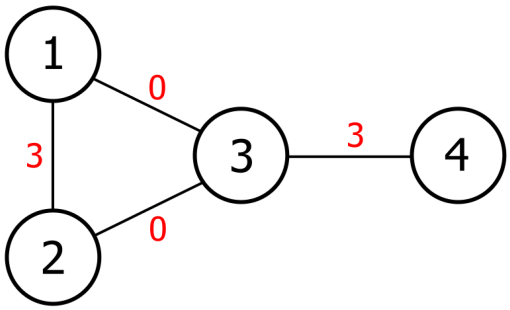


Figure 3: Uneven distribution of bandwidth

In Figure 2 node 3 distributes his bandwidth evenly to all links. Nodes 1 and 2 use their remaining bandwidth for their direct connection, node 4 can not reach its maximum. This is exactly the balancing ALTRUISTIC and RATE use. Once they get in a stable state, they do not change the speed of the links any more. That's why these algorithms lead to a fast stabilization of the average download speeds, while the rates in BITTORRENT and DEFICIT jump up and down and the simulation needs a long time to create stable average download speeds.

In Figure 3 we have a better resource allocation. Node 3 communicates only with node 4 and leaves the links to nodes 1 and 2 unused. This seems not to be 'fair' for nodes 1 and 2, but is much better regarding the global optimum. Every node in the example above has a maximum utilization of its connection. This behavior can be observed in BITTORRENT and DEFICIT and is highly dynamic as the links are opened and closed again and again.

5.9 Quality of Results

5.9.1 Discussion

With this simulation I only was able to simulate the real world omitting many details. Because no real TCP connection are made, network links are too stable and newly opened connection get too fast to full speed. Especially those protocols with frequent speed changes (DEFICIT and BITTORRENT) perform too well in the simulation.

Another point is the omitted fragment scheduling in the simulation. In reality not all desired fragments are available to a node of the system, meaning a worse

performance. In the simulation a node can download from its fastest peers assuming they provide the right chunks. We get one result from Jun and Ahamad's simulation on PlanetLab, in which BITTORRENT shows a more efficient fragment scheduling than DEFICIT because of the less restrictive neighbour selection.

It is difficult to compare my results with other simulations because of the different performance measurements. Jun and Ahamad, for example, look at the completion time for a sample file. This gives a more detailed view on the dependence of the completion time on the upload rate. Surprisingly all peers running BITTORRENT finish at about the same time while with DEFICIT the completion time depends on the upload rate. Fast upload rate means fast completion time, but the mean completion time of DEFICIT is bigger than the one of BITTORRENT, because of the less efficient fragment scheduling. The main conclusion, that we can get a better fairness than BITTORRENT without much performance loss can be confirmed.

Unique in my simulation is the influence of the free rider ratio to the system and the analysis of the robustness of the algorithms to this ratio.

Although I used only a simple simulation, I was able to show the main behaviour of the different algorithms concerning performance and fairness.

5.9.2 Optimal Performance

I used ALTRUISTIC to get a performance reference. As described in Section 5.7.2, 100% performance means about 82 % of the possible download speed. This looks quite arbitrary and a more precise definition of maximum performance is needed.

This optimisation problem can be transformed into a maximum flow problem. We consider each node of the system as a source and a sink. Because a node can not download from itself and the download speed only depends on the direct neighbours in the graph, we get the following transformation of our toy network from Section 5.8:

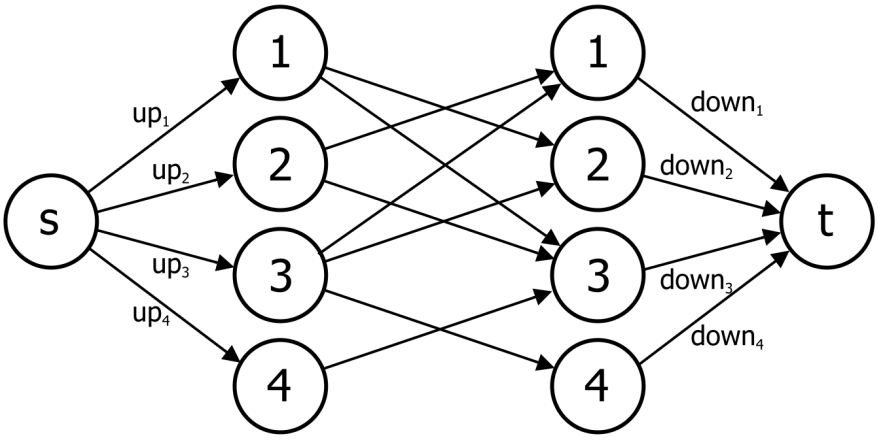


Figure 4: Flow Graph

The capacities of the edges leaving the source are set the upload rates of the corresponding nodes. The capacities of the ones entering the sink equal the download rate of the nodes. All other edges have infinite capacity.

This problem can now be formulated as a linear programming model and can for example be solved by simplex algorithm.

6 Future Work

My improvement on DEFICIT by introducing the RATE algorithm did not work in this first attempt. I still think it would be possible to increase the performance of RATE to get a more constant alternative to DEFICIT without losing much fairness. This could be done with a better connection management as explained in Section 5.8. The same applies to ALTRUISTIC, which uses a similar connection management than RATE.

We have seen that good algorithms group nodes of equal link bandwidth to increase overall performance. This leads to the open question, how fairness exactly influences performance. Is there something like a ‚price of mechanism' or do we have to speak about ‚benefit of mechanism'?

Apart from the improvements on the algorithms, one could refine the simulator to distinguish between different file chunks. This would give a more detailed view how chunk selection relates to fairness and performance. Another improvement on the simulation would be using a more realistic network e.g. PlanetLab [16].

We have seen that tit-for-tat seems to be a very good approach for ensuring a high fairness for file sharing systems with many users and few files, like in BitTorrent. The cooperation of a node directly influences its utility. In traditional file sharing systems a node perhaps is never rewarded for its upload to a peer, because it is not interested in the offer of this peer. The same applies to all other systems without pair wise utility exchange, for example file storage systems. Is it possible to define something like a global tit-for-tat or a persistent, distributed fairness system, which stores peer contribution information?

7 Conclusion

7.1 General Conclusion

If a system does not provide a mechanism to prevent people from using it without contributing anything, few nodes have to handle a big amount of the load. That's why it is important to introduce a simple, but effective mechanism to control the users' activity. The only possibility for a node is to control its direct neighbours, all other attempts in the past failed.

A good algorithm does not only have to be fair, it also has to be fast. This can be seen as a global resource allocation problem. A good algorithm should not end up in a local maximum. This point still remains open and is subject for further studies.

We have seen that mechanism design is an important part of designing a distributed system. It creates incentives for selfish users to contribute to it and influences both fairness and performance. If tit-for-tat is really the only promising strategy and how existing algorithms can be improved will show us the future.

7.2 Personal Conclusion

As the first part of the task description for this thesis was very precise, it was clearly what to do, namely to compare the different existing mechanism designs. It was interesting for me to see how those systems try to lead users into cooperation and that some system are used today by millions of people and still existing, even if their mechanism design is inexistent or useless. If I look back on this first part, I state that I spent too much time, which I would have needed in the second part.

The focus of the second part was left open and I have had several possibilities:

- Modifying a Gnutella or BitTorrent client for free riding
- Solve theoretical questions
- Write my own simulator and compare different algorithms

Writing a Gnutella Client would have been senseless because no countermeasures are used in this network and no interesting questions could have been answered. Doing the same in the BitTorrent network would have been much more interesting. I have not done this because of two reasons: First, the investigated client Azureus has shown that the BitTorrent protocol leads to complex clients and modifying Azureus would have exceeded the work of this thesis. Second, this would have been only an 'attack' and I would have been unable to defend it. As the first part has been more of analysis nature, I wanted to do in the second part something more productive. This is why I left the theoretical questions behind und started writing the Java simulator.

I'm not completely satisfied with the results of the simulator, mainly the performance of ALTRUISTIC and RATE are discontenting for me. Apart from that, it was a pleasure for me to write this simulator and I have spent much time looking at the algorithms running in GUI mode.

8 References

[1] Wikipedia, http://wikipedia.org/

[2] Schweizerisches Urheberrechtsgesetz, Art. 19
Stand: 23. März 2004

[3] O. Heckmann, J. Schmitt, R. Steinmetz.
Peer-to-Peer Tauschbörsen – Eine Protokollübersicht
http://disco.informatik.uni-kl.de/publications/HSS02-3.pdf

[4] E. Adar und B. A. Huberman. Free Riding on Gnutella.
First Monday Peer-Reviewed Journal on the Internet, 5(10), Oktober 2000.
http://www.firstmonday.dk/issues/issue5_10/adar/index.html

[5] Slyck, http://www.slyck.com/

[6] Freenet FAQ
http://freenet.sourceforge.net/index.php?page=faq

[7] Mojo Nation - sicheres P2P Filesharing
Online Article, http://p2p.at-web.de/mojonation.htm

[8] S. Krempl. Mojo Nation: Die ultimative Mischung aus Napster und eBay?
Online Article, http://www.heise.de/tp/r4/artikel/8/8466/1.html

[9] Y. Kulbak, D. Bickson. The eMule Protocol Specification
http://www.cs.huji.ac.il/labs/danss/presentations/emule.pdf

[10] Kazaa Participation Level
http://www.kazaa.com/us/help/glossary/participation_ratio.htm

[11] Game Theory Definition
http://www.gametheory.net/Dictionary/ParetoEfficient.html

[12] B. Cohen. Incentives Build Robustness in BitTorrent
In Proc. of the First Workshop on Economics of Peer-to-Peer Systems, 2003
http://www.bittorrent.com/bittorrentecon.pdf

[13] R. Axelrod. The Evolution of Cooperation. Basic Books, 1984.

[14] D. Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-Like
Peer-to-Peer Networks. In Proc. ACM SIGCOMM, 2004.

[15] S. Jun and M. Ahamad. Incentives in BitTorrent Induce Free Riding
ACM SIGCOMM '05

[16] PlanetLab
http://www.planet-lab.org/