The Open University of Israel
Department of Mathematics and Computer Science

FairE9: Fair File Distribution over Mesh-Only Peer-to-Peer

Thesis submitted as partial fulfillment of the requirements
towards an M.Sc. degree in Computer Science
The Open University of Israel
Computer Science Division

By
Eyal Zohar

Prepared under the supervision of Dr. Anat Lerner

January 2009

Acknowledgments

I wish to thank Dr. Anat Lerner for her guidance and endless patience; Prof. Danny Raz for his productive advice; Prof. Zeev Nutov for helping with the permutations algorithm; Ran Tavory for his comments on earlier drafts; and Ofer Wald for recommending me to write in English.

Table of Contents

Tables and Figures

Abstract

Peer-to-peer (P2P) networks are in the spotlight due to the wide-spreading file-sharing applications. Many file-distributing algorithms have been suggested and implemented. The various solutions need to cope with a heterogeneous and unstable environment, where peers can arrive and depart at a high rate (churn). Sometimes cooperation cannot be assumed. These issues make the structured attitude less practical. Even some of the algorithms that are considered as unstructured try to maintain long-term parent-child relationships. Existing unstructured (mesh-only) algorithms for file-distribution work well on the Internet on the average. But some of the participating peers may suffer from a slow start or high latency because of the randomness of the peer and piece selection for upload and download.

In this paper we propose a fair unstructured system for file-distribution from a single source, with no central authority. The proposed protocol is fair both with respect to load balancing and with respect to the latency in each peer. It is based on a novel weights-algorithm that helps peers to determine what piece to ask from which peer, in a manner that increases their chance to get served. In this way it also lowers the overhead. The proposed algorithm welcomes newcomers while being resilient to churn, being resilient to free-riders, and adaptive to heterogeneous bandwidth.

1. Introduction

The peer-to-peer (P2P) system is a promising platform for many forthcoming distributed systems that deal with content delivery. The magic of P2P systems is based on the ability of a client to share previously downloaded data, thus helping numerous servers with their distribution. Broadcasting servers can deal with a large client-base by having the clients form a P2P overlay. With such an overlay, web sites and multimedia servers can cope with a sudden increase in service demand (flash-crowds) [1].

P2P overlays can be classified into two main classes: mesh-only and non-mesh. The non-mesh solutions are mostly based on some (hidden) hierarchical structures, with long-term parent-child relationships. Initial effort is spent on the establishment of a structure, which in turn allows an efficient streaming of the information along the paths.

SplitStream [2] is a tree-based multicast system for a heterogeneous environment. It splits the video stream into stripes using MDC (Multiple Description Coding). With MDC, a peer can watch the video as soon as it receives at least one stripe, but to

improve its quality it needs more stripes. SplitStream distributes each stripe using a separate distribution tree. If the forest is built efficiently, all the peers share the forwarding load. The forest structure also takes into account the *declared* capacities of every peer to maximize the peer utilization. It relies on Pastry [3] to maintain the distribution tree without the need for a central authority.

Many existing solutions follow the idea of SplitStream to build multiple trees/paths to distribute the forwarding load among all peers. Each algorithm may focus on a different property. For example, Orchard [4] deals with free-riding, Chunkyspread [5] adapts to heterogeneous capacity, LagOver [6] considers individual latency constraints, and Climber [7] and [8] are incentives-based.

These structures can be very efficient in a cooperative and stable environment, where peers stay for long-term and contribute a stable and significant upload bandwidth. But these algorithms often lose their main advantage in the Internet environment, where peers arrive and leave (churn) frequently, and where some are unwilling or unable to share in return (free-riding). Broken paths are not easily detected, and when these are repaired they cost with some more overhead and a higher latency.

Mesh-only solutions divide files into small equal pieces to allow peers to upload soon after they start download. The peers are given a high level of freedom, as they can contact any peer, and download from it any piece the other peer owns and is willing to upload. Central entities, if they exist, do not fully control the process but merely help peers to locate each other. Famous representatives of this family are the existing file-sharing networks, like BitTorrent [9], Gnutella [10] and eMule [11]. These operate well in a heterogeneous bandwidth environment, and sometimes encourage peers to share. But their flexibility and openness come with some level of chaos which results in random performance and encourages peers to be aggressive by contacting many siblings. The distributed nature and direct-reciprocity help free-riders to bypass defense mechanisms by whitewashing (switching identity) and/or downloading from complete-sources (peers that completed their download) [12]. Mechanisms that are intended to filter out free-riders may also hurt potentially-cooperative newcomers [13].

In this paper we suggest a novel approach to the problem, which puts some order in a mesh-only solution. Each peer has an inherent individual piece-order that can be easily derived by all other peers from a commonly known function. The combination of the two peers' orders imposes weights to each request, helping to estimate the probability for a transaction before the requests are actually sent. Altogether this algorithm

enhances fairness and lowers the overhead of the process.

This paper is organized as follows. Some background is given in Section 2. The proposed algorithm is presented in Section 3. In Section 4 we show experimental evaluations. We present several ideas for future work in Section 5 and conclude in Section 6.

2. Background

FairE9 takes a novel approach toward the file distribution problem. To our knowledge, there are no other mesh-only P2P networks that are designed to cope with flash-crowd. In this section we give a brief overview of some systems that will later be referenced and compared to.

2.1. BitTorrent

Direct reciprocity P2P networks are based on the concept of direct reward given by a downloader to a peer that uploaded to it. This may be an immediate and short-term reward, or a future score that will help the generous contributor to get better service later when required. Such schemes tend to be simple to understand and implement in practice, and are therefore popular among the current P2P file sharing systems.

BitTorrent is a pioneer of mesh-only P2P file-sharing, which can be regarded as having a direct-reciprocity mechanism with an immediate reward. The implemented mechanism is roughly based on TFT (Tit-for-Tat) [14]. Studies show effective upload bandwidth utilization [15]; however, it is also far from optimal for distribution from a single source to multiple receivers [16]. In addition, its peer selection policy induces free-riding [17], and measurements show that when the number of seeds is over 60%, free-riders download faster than others [12].

The distribution process is made up of many peers running a software client that implements the original BitTorrent wire protocol. Numerous software clients exist, each has its own flavor: from a basic open-source, to sophisticated clients that use their proprietary extensions. The incentive mechanism is not a part of the wire protocol, but is more related to internal decision taking. As with the wire protocol, each software client may take a slightly different approach.

Basically, each peer holds a fixed number of upload slots, hopefully allocated to cooperative and beneficial partners. Peers normally measure their download speed in these slots, in order to choke (send a specific "choke" message) or even disconnect the less productive ones, and look for better partners. In order for cooperation to get

started, it also does an "optimistic unchoke" from time to time, meaning it gives a new partner an upload slot for a short time to see if it cooperates. This behavior is intended for newcomers to get their time-limited chance to get their first pieces, but it also helps free-riders to download.

2.2. eMule

eMule file sharing system was suggested earlier than BitTorrent. It is considered less efficient than BitTorrent, and it is less popular worldwide today. It implements a long-term credit system, in which peer a gains a remote credit when uploading to peer b, and loses some credit when downloading from it. If peer a later needs anything that b shares, it will move faster in b's queue and get better service according to its credit there.

eMule's main drawback is long latency and large overhead. Each peer has a single incoming-requests queue that is common to all the files it shares. Peers are encouraged to contact many potential sources and to send multiple messages just to keep their place in the remote queues until they get a chance to download one piece (around 10 MB). Sometimes it takes several days before a peer gets a significant part of a downloaded file.

3. The FairE9 Approach

3.1. The Model

A group of peers $N = \{1,...,n\}$ is interested in a file $f \in F$ [1]. The file is divided into m pieces, indexed with serial numbers $\{1,...,m\}$. Initially the file is held exclusively by the *initial source* $s \in N$. The goal is to distribute all the pieces to all the peers in N. When a peer completes downloading all the pieces of the file, it is called a *seeder*.

3.2. The Idea

Every peer has a strict piece-order, which is a function of its identity (e.g. network address) and the file identifier (e.g. file hash). The permutation function g maps each triple of a peer, a file, and a piece number into the index in the peer's piece-order. It involves a procedure that converts a number to permutation, called "unranking a permutation". These papers show how to perform it in a linear time [18;19]. The function g is a bijection (one-to-one and onto), and it is known to all peers in N:

$$g : \left[N, F, \{1,...,m\} \right] \to \{1,...,m\} \qquad (1)$$

[1] FairE9 supports the distribution of multiple files in parallel, while each is distributed in a separate domain, like BitTorrent does.

Each peer repeatedly sends requests to other peers, asking to download a specific missing piece. Sources get these requests, and consider providing the needed piece. Each request is granted a calculated *weight*, such that the requests with the *lower* weights get higher priority and are served first.

Weights are based on piece position on both sides. When peer con asks for piece i of file f from peer src, both peers measure the request's weight using the formula:

$$weight(con,src,f,i) = \begin{cases} g(con,f,i)+g(src,f,i) & if\ src \neq s \\ g(con,f,i) & if\ src = s \end{cases} \quad (2)$$

3.3. Peer Identifier

The permutation function makes use of the peer identifier. A reasonable peer identifier mechanism would be the IP address of a peer, converted to a serial integer (up to 2^{16}). Other alternatives may be considered, as long as they follow the basic required properties of such an identifier. The identity should not be freely selected by the peer itself, although non-frequent switches are allowed. FairE9 does not save any credits, therefore multiple peers with the same identifier may exist, just not too often; groups of around $1/m$ of the peers can share the same identifier with no harm.

3.4. Locating Peers

FairE9 does not limit itself to one solution, but it can use any combination of one of the many operative solutions already existing in current networks; for example BitTorrent trackers, BitTorrent DHT, eMule servers, eMuleKademlia, etc. For the ease of description, from this point on, we assume that a BitTorrent trackers mechanism is in use.

3.5. Random Protocol

In this section we describe a fully random algorithm that can be considered as the ancestor of several mesh P2P networks, like eMule, BitTorrent and FairE9.

Procedure 1 ***MainRandom***

1. Once: *Bootstrap*()
2. Forever, in background: *PeerExchange()*
3. Forever, in background: *AcceptRequests()*
4. Until seed, in background: *SendRequests()*
5. Forever: *Upload()*

Procedure 2 ***Bootstrap***

1. Get initial peers from external source
2. Add peers to *known_peers*
3. Add peers to *free_peers*

Procedure 3 ***PeerExchange***

1. Get random peer from *known_peers*
2. Ask peer for its *known_peers*
3. For each returned peer, while not exceed *known_peers*:
   3.1. If returned is new to *known_peers*:
      3.1.1. Add returned to *known_peers*
      3.1.2. Add returned to *free_peers*

Procedure 4 ***AcceptRequests***

1. Wait for incoming requests
2. If not exceed *up_pending*:
   2.1. Add request for random piece to *up_pending*

Procedure 5 ***SendRequests***

1. Cleanup expired *down_pending*
2. Remove disconnected *known_peers*/*free_peers*
3. If not exceed *down_pending*:
   3.1. Pop random peer from *free_peers*
   3.2. Send request for random piece to the peer
   3.3. Add the request to *down_pending*

Procedure 6 ***Upload***

1. If *up_pending* is not empty and *up_slots* is not full:
   1.1. Pop random peer from *up_pending*
   1.2. If there are pieces you have and peer does not:
      1.2.1. Pick random piece of these pieces
      1.2.2. Occupy a slot in *up_slots*
      1.2.3. Upload the piece to the peer in background
      1.2.4. When complete, release a slot in *up_slots*

3.6. FairE9 Protocol

FairE9 suggests a different approach to sending requests and to uploading:

Procedure 5a ***SendRequests***

1. Cleanup expired *down_pending*, and abandon the peers that were attached to the removed requests
2. Remove disconnected *known_peers*/*free_peers*
3. If not exceed *down_pending*:
   3.1. *target*[2]←max(2,*down_pending* / missing pieces)
   3.2. Get the lightest missing piece that has less than *target* pending requests
   3.3. Pop from *free_peers* the peer with the lightest weight for that piece
   3.4. Send the request for the piece to the peer
   3.5. Add the request to *down_pending*

[2] This formula is used in the simulator, later in this article. Other formulas may be used instead, as part of the trade-off between speed and overhead. This subject is out of the scope of this paper.

Procedure 6a ***Upload***

1. If *up_pending* is not empty and *up_slots* is not full:
   1.1. Pop the lightest peer and piece from *up_pending*
   1.2. If you have that piece:
      1.2.1. Occupy a slot in *up_slots*
      1.2.2. Upload the piece to the peer in background
      1.2.3. When complete, release a slot in *up_slots*

3.7. Expected Behavior

The idea behind the weight formula is to advise peers how to behave. Downloading peers are expected to send download requests only to peers from whom there is a high probability of being served. Each request specifies the specific missing piece with the lowest weight. This mechanism has several fundamental desirable properties: it welcomes newcomers in the sense that new peers will get their first pieces very fast; the total overhead is reduced as the possibility of being served is increased; and the total load is more fairly shared as peers will usually receive requests for their low-order pieces. Requesting peers temporarily abandon peers that have not uploaded to them fast enough (expired *down_pending*), in order to cope with free-riders and slow peers.

A detailed example is illustrated in Figure 1. In this example, a group of $n = 28$ peers distributes a file f of $m = 6$ pieces. The illustration is for one of the pieces only - piece i. For the ease of the demonstration, we assume synchronous time, and we demonstrate 3 imaginary time-slot phases that resemble only one of the many possibilities to distribute a single piece. We further assume that each peer may know only a subset of the peers; each peer may perform several uploads and downloads in parallel; other pieces may be transferred in the background (not displayed in this illustration).

Figure 1a shows 3 peers downloading piece i from the initial source s. Their given piece-order index is drawn in the upper-right corner of each peer, meaning that:

$$g\left(k_2, f, i\right) = 1$$
$$g\left(k_3, f, i\right) = 1$$
$$g\left(k_4, f, i\right) = 2$$

Figure 1b is a later possible snapshot. Note that k_6 downloads after k_4, although $g(k_6,f,i) < g(k_4,f,i)$, because k_6 did not know s at the time but it knows k_2.

Figure 1c shows the last downloads of this piece. Note that unlike a classic tree, here s,k_2,k_3 can upload again, if they have free upload bandwidth; k_9,k_{10} do not upload this piece, because of a relatively high weight compared to $k_6,k_7,k_8,k_{11},k_{12}$; k_5 has the same high weight but it uploads, maybe because k_{15} does not know many other peers or because k_{15} did not manage to download the piece from others yet, due to others' bandwidth constraints.

Note that the other $m-1$ pieces are distributed in parallel in the background, involving the same peers. For each piece, a different tree-like distribution is performed. In each tree, different peers carry the load. This outcome resembles the idea behind SplitStream, but without any structure, no central authority, no need for full knowledge of all the peers, and resilience to peer failures and noncooperation.

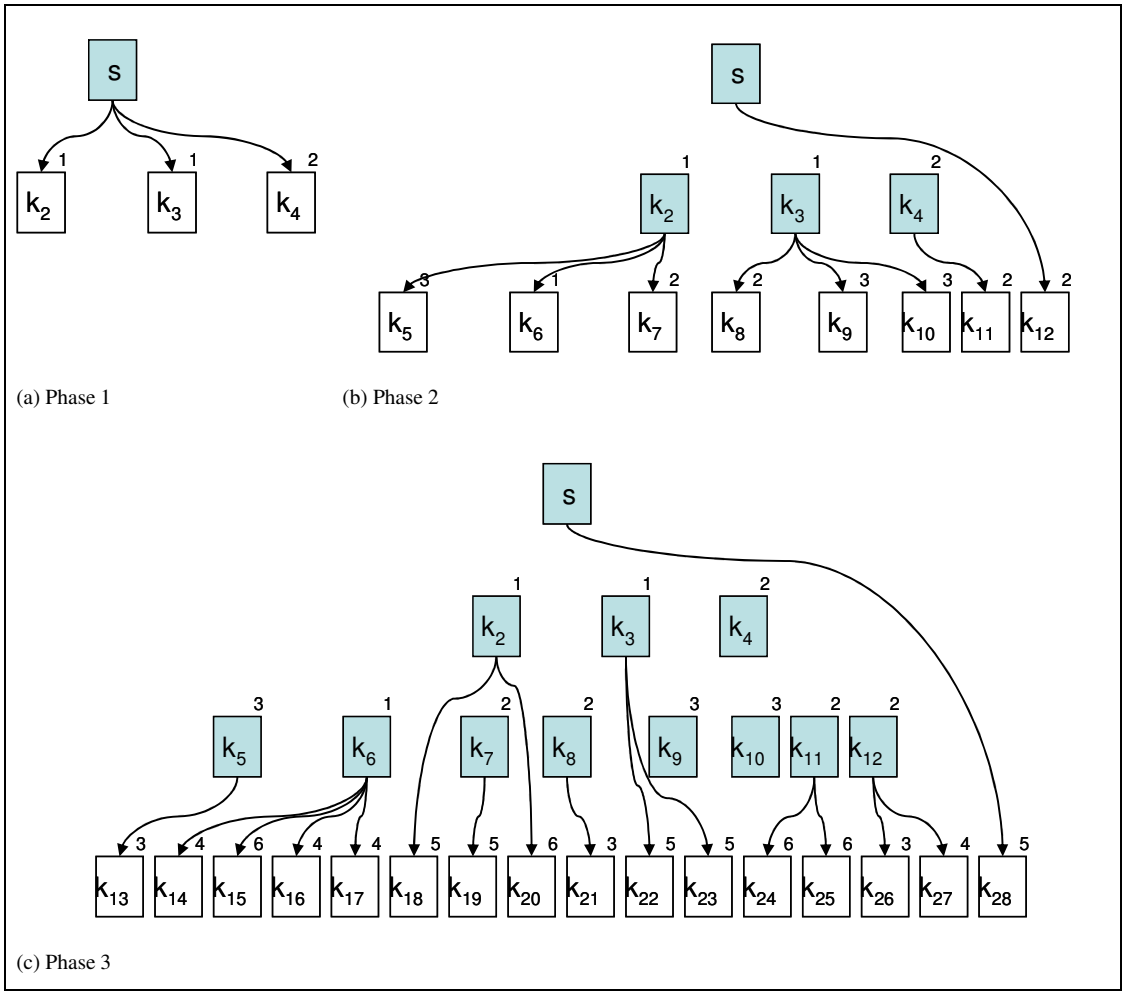


Figure 1 Tree-view of a single piece distribution

4. Experimental Evaluation

In this section we present simulation results of experiments designed to evaluate the performance of FairE9 and compare them to existing algorithms. We ran large-scale experiments on a proprietary network simulator that implements several different systems in an almost real environment. The results support our claims that several aspects of fairness and overhead can be improved by the new system.

4.1. Simulation Setup

We developed a proprietary simulator in order to evaluate specific behaviors of the proposed system, while neutralizing mechanisms that are not in the scope of this paper, such as the peers acquiring mechanism.

All the simulated systems are evaluated using the same fixed parameters, unless noted otherwise: 10,000 peers download a 250MB file, with 200 pieces, from a single source; each peer, including the single source, has a 3Mbps download and 3Mbps upload bandwidths, divided into 500Kbps slots; each peer has 5 initial random known peers (previously joined), and performs a single peer exchange with one of the peers it knows every 40 seconds to get up to 50 new peers, until it reaches 500 known peers; maximum number of pending requests is set to 100, with timeout of 200 seconds per request.

To simulate eMule, we created an eMule-like mode that resembles Random, but uses eMule credits mechanism. Peers waiting in the queue get higher priority if they have uploaded to the same peer before. Each uploaded piece gives a boost equal to a 60 seconds wait in the queue.

BitTorrent does not have a strict united incentives mechanism because many software clients exist today, each taking a slightly different approach. Our BitTorrent-like mode implements the basic ideas presented by the original software client [20]; each peer regularly checks the download-to-upload ratio on each connection to keep the connections with the high ratios alive, and closes those with the lower ratios. It also periodically gives a chance to new peers for a limited time.

4.2. Flash Crowd and Fairness

We tested an extreme case of flash crowd, when the single source deals with all the newcomers simultaneously from the very beginning. This is a common situation with live events or timed site launches. The goal is to keep both the average latency (efficiency) and the maximum and standard deviation latencies (fairness) low.

This is a difficult assignment for mesh P2P that deals at start with peers that know only some of the other peers; none of them have a history or anything to give in return. We wish to avoid unfair situations, where some peers download several pieces while others starve for their first piece.

In Table 1 we compare Random, eMule and FairE9, with respect to the average, the standard deviation and the maximum latencies. The simulation results show that FairE9 is both more efficient and more fair than Random, eMule and BitTorrent.

Protocol	Avg. Latency	Latency Stdev	Max Latency
Random	795.4 sec	35.6	1,100 sec
eMule	812.8 sec	38.8	1,260 sec
BitTorrent	819.2 sec	47.6	1,300 sec
FairE9	786.4 sec	19.8	880 sec

Table 1 Flash crowd results

To explain the differences, a closer look at the process is needed. Figure 2 presents the percentage of slower peers that were at least 20 pieces (10% of the file) behind the average. Figure 3 does the same for 40 pieces (20%) behind the average. The diagrams show the undesired impact of BitTorrent's TFT and eMule's credit system, where a relatively small group of peers is starved. These are unlucky peers that get their first pieces late; therefore they have lower credit and slow continuation, even in comparison to Random. FairE9 on the other hand helps peers that started late to catch up quickly.

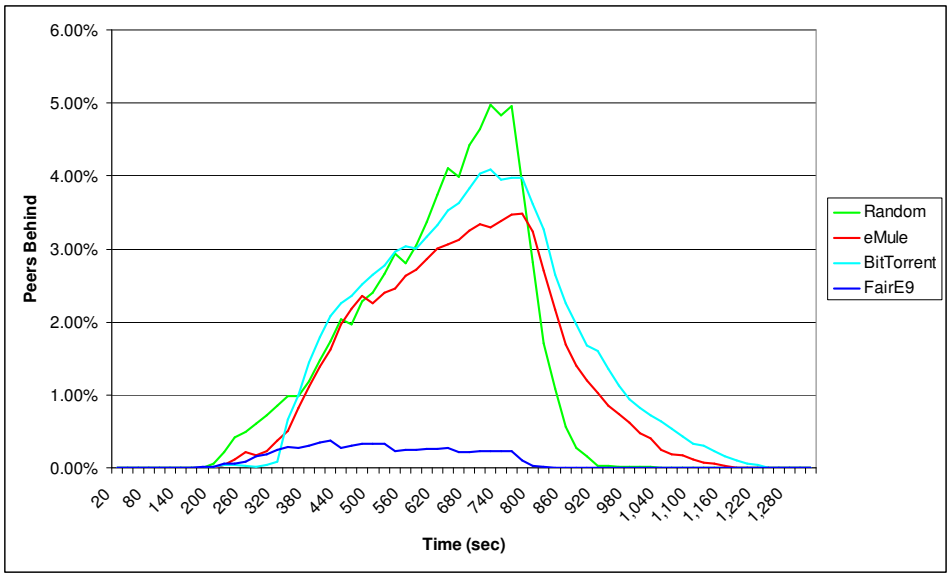


Figure 2 Peers 10% behind the average

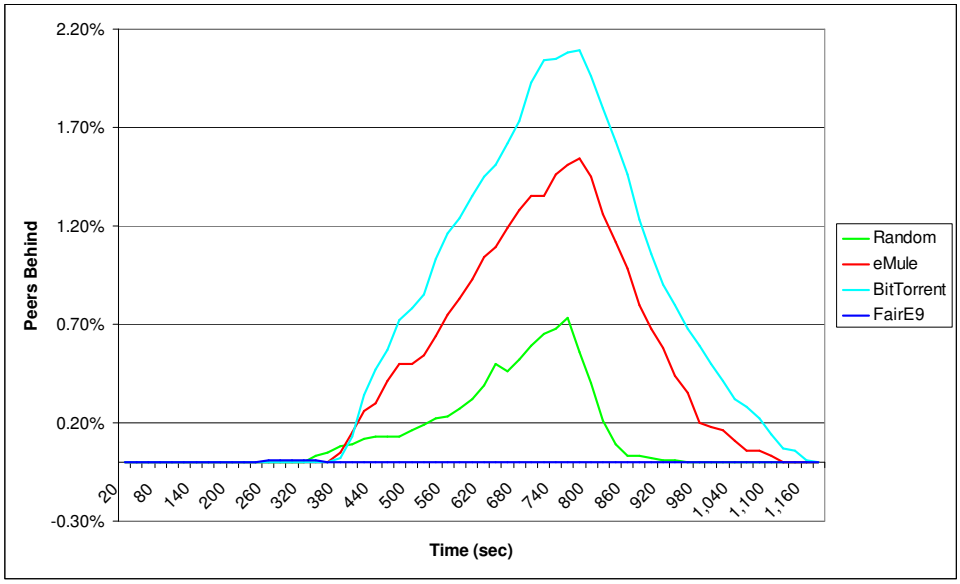


Figure 3 Peers 20% behind the average

4.3. Load Balance and Fairness

Another aspect of fairness is related to the load carried by each peer. We have measured the number of uploads performed by each peer. The results in Table 2 show that FairE9 keeps the system relatively balanced, probably because it turns newcomers into "almost seeders" very fast, as they get their first pieces early, and these first pieces are also the pieces often requested from them by others.

Protocol	Uploads Stdev.	Max Uploads
Random	29.52	288
eMule	26.01	339
BitTorrent	31.94	358
FairE9	20.95	253

Table 2 Uploads deviation

BitTorrent demonstrates the worst results, probably because some of the peers are unlucky with their first pieces, and TFT only makes the situation worse. Not only do peers not get a chance to upload much, they also need to download mostly from seeders that do not ask for anything in return. Altogether it creates an unbalanced load.

4.4. Moderateness and Message Overhead

Typical mesh P2P encourages peers in practice to contact many peers, as the chance to download grows with the number of requests sent. What stops peers from sending unlimited number of requests are costs and constraints related to communications and CPUusage.

Consider for example a peer that sends i requests. Denote by p the probability to be served by at least one of the requested peers. Using the Random algorithm, if this peer

doubles the number of requests, the probability to be served will grow to $2p - p^2$, as the probability to download from the peers of the second group is equal to the probability to download from the peers of the first group. With FairE9, the requests are ordered in a descending order of the probability to be served. Therefore, if we denote by q the probability to download from the second group, then $q \ll p$, and so doubling the number of requests will be less profitable.

To evaluate the motivation of a single peer in practice, we defined a peer *aggressiveness factor* which is a multiplier of the number of maximum pending requests (the base is 100). Table 3 shows what happens when the crowd has an aggressive factor of 2.0 while a tested group of 10 peers remains with a factor of 1.0. With respect to the reduced latency of the crowd and the added latency of the group, it is clear that Random is the worst. One could expect that the moderate group would not perform so well in eMule and BitTorrent because their somewhat random service encourages peers to be aggressive. But surprisingly the moderate group performs well in both networks because in this special experiment there are hardly any seeders during the process which makes the direct-reciprocity more dominant and random contacts less effective. The reason that FairE9 is influenced by the factor is because the aggressive peers have an advantage when downloading their last pieces.

Protocol	**Crowd avg latency reduction (1.0→2.0)**	**Group avg latency gain (2.0→1.0)**
Random	18.2 sec	154 sec
eMule	0.6 sec	10 sec
BitTorrent	2.4 sec	6 sec
FairE9	12.2 sec	30 sec

Table 3 Aggressive crowd vs. moderate group

Message overhead has some performance penalty, therefore in real life there is a trade-off between the reduced latency and the additional overhead. Therefore, we also measured the average number of messages per download sent by the peers. The results are presented in Table 4. Intuitively, as FairE9 sends messages for specific pieces while the others ask for any piece, the number of messages in FairE9 could be the highest. But, in the others, the peers have to exchange inventory messages to figure out if and what one can send to the other, so the final results are that FairE9 has a lower message overhead in this test case. BitTorrent has the highest overhead, probably because the unlucky newcomers need to send many messages until they get a chance to download. eMule is a little better, probably because peers mostly get a chance to download even

without a credit, just by waiting long enough for their turn.

Protocol	Messages per download
Random	5.15
eMule	5.82
BitTorrent	6.53
FairE9	4.82

Table 4 Message overhead

4.5. Welcoming Newcomers

A common problem with incentives-based systems is the way they treat the newcomers who cannot contribute anything or cannot express or prove their willingness to share.

To emphasize the effect of the approach taken by FairE9, we simulated a crowd that joins the distribution gradually; 8,000 peers join at time 0, and 200 more join every minute, up to 10,000 peers. For each group of 200 peers we measured the average latency and absolute completion time (join time plus latency). We found that the absolute completion time correlates with the join time on all the protocols and groups. But the latency results are more complex, as presented in Figure 4.

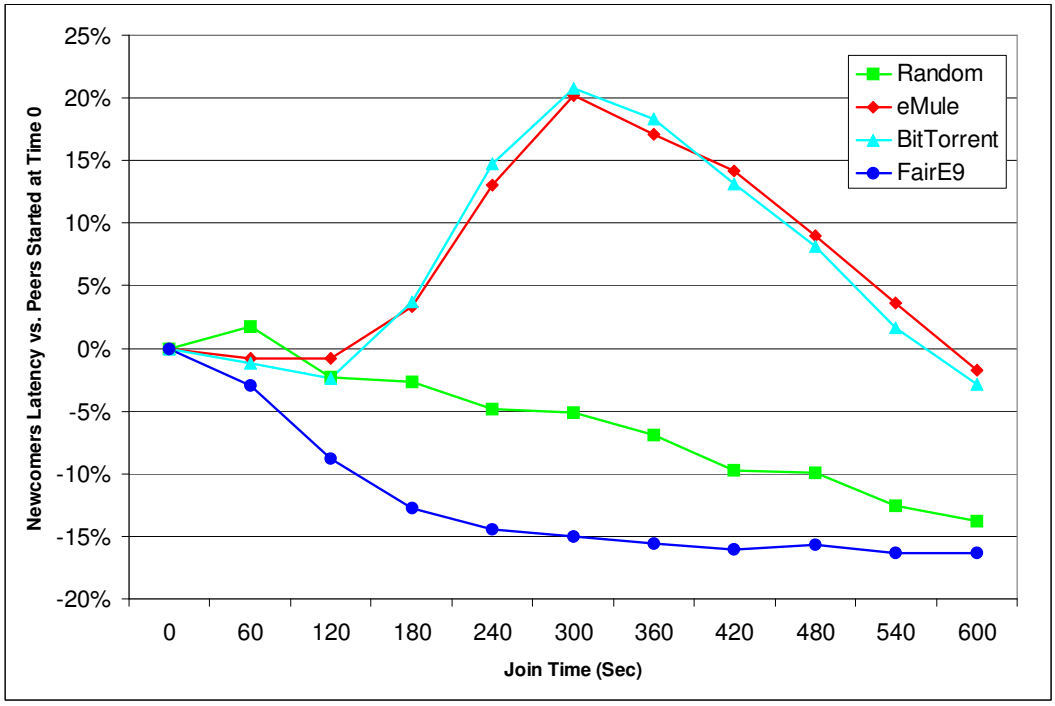


Figure 4 Newcomers latency

Newcomers in eMule and BitTorrent usually pay for their late join by having a long latency. An exception to this is when they join the distribution in a progressed stage, with a high number of seeders (or almost seeders).

We also compared the delay for receiving the first piece by the newcomers, as shown in Figure 5. The samples are taken one minute after a group of 200 peers joins, just before another group joins. It is clear that FairE9 and Random help newcomers to download, as almost all the 200 peers download at least one piece within the first minute that they join. eMule and BitTorrent incentive mechanisms treat newcomers as inferior peers, therefore they have difficulties getting their first piece.

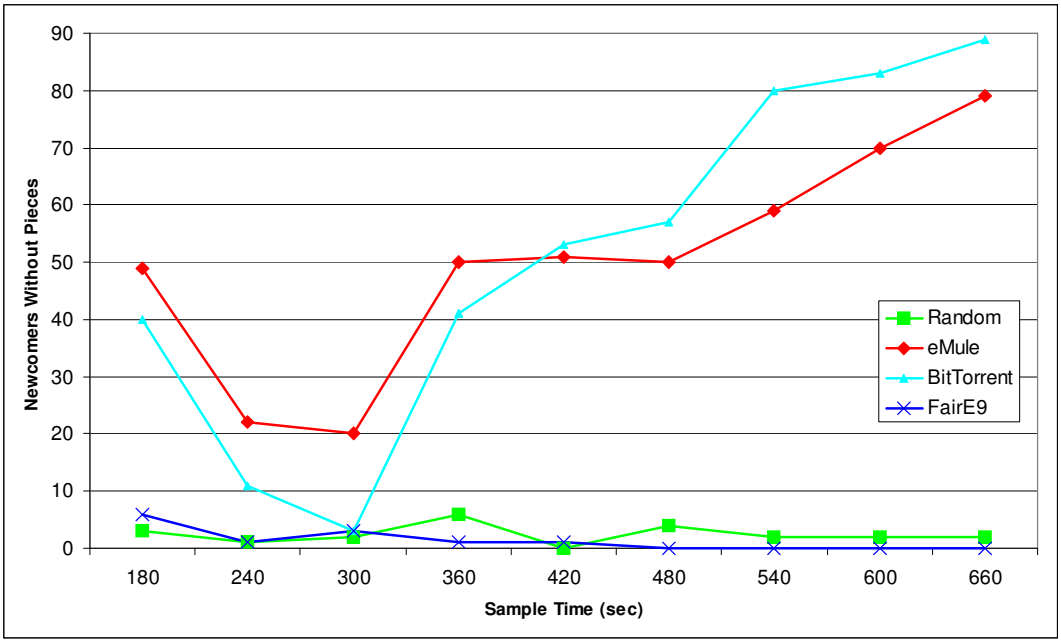
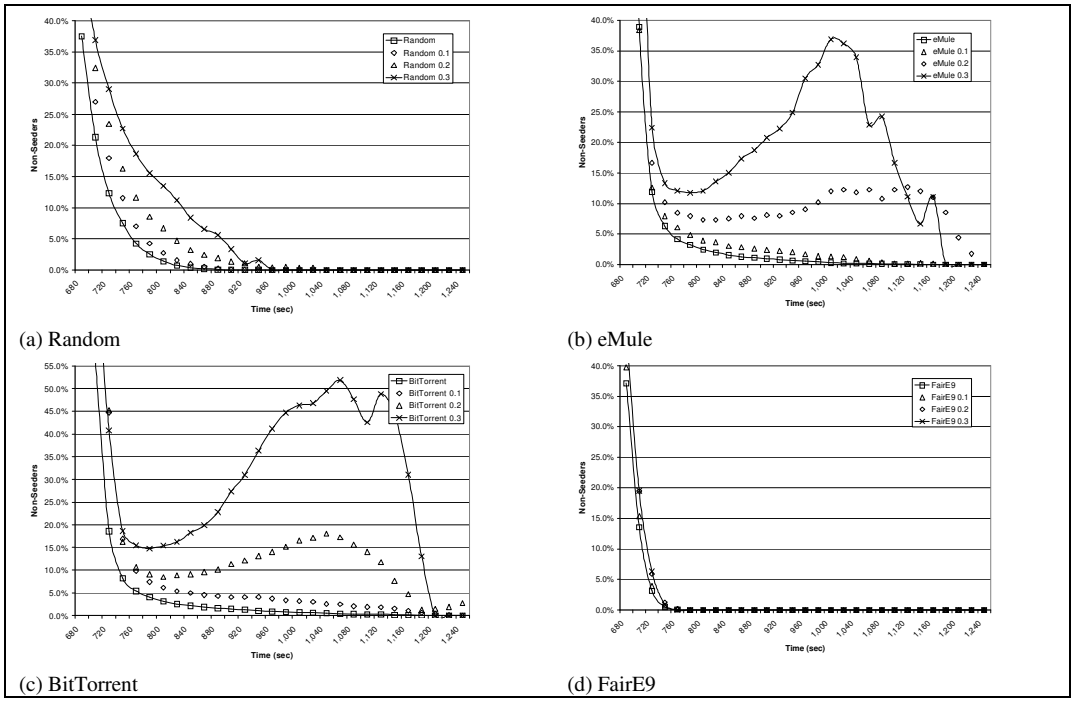


Figure 5 Sampled newcomers

4.6. Resilience to Seeders Churn

The problem with peers that leave unstructured systems is that they might take with them rare pieces of the file or just take with them a significant part of it while the overall progress is just at the beginning.

We have tested the case of peers that leave soon after they become seeders. Figure 6 presents the portion of non-seeders still in the system, while seeders leave with probability 0, 0.1, 0.2 or 0.3 every 20 seconds. The results correlate with Figure 3, because eMule's starving peers also suffer from the phenomenon explored here, while FairE9 is completely resilient even to a high churn rate.



(a) Random (b) eMule

(c) BitTorrent (d) FairE9

Figure 6 Seeders churn, per protocol

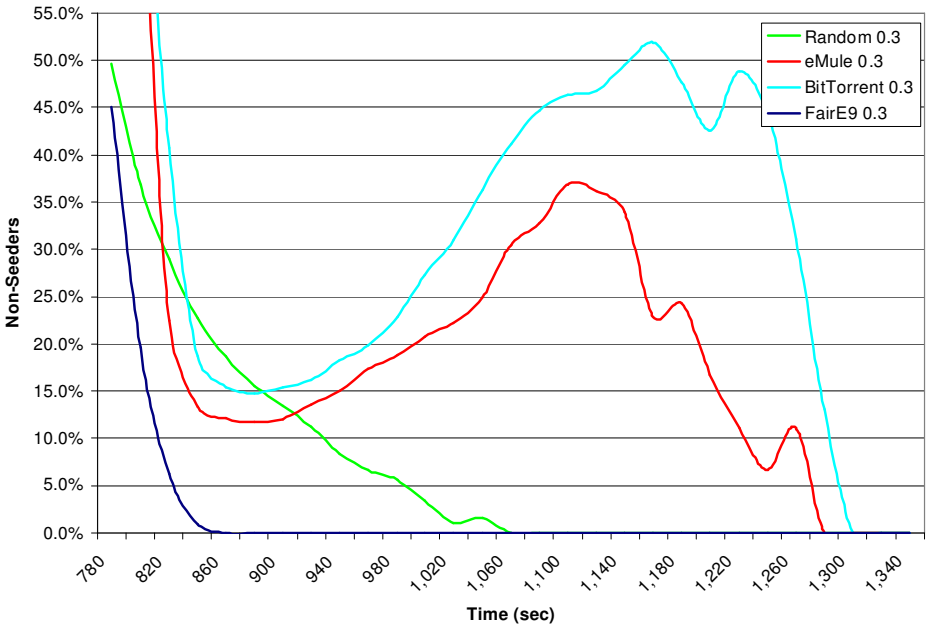


Figure 7 Seeders churn, with leaving probability 0.3

4.7. Incentives to Share

Any comparison that involves eMule and BitTorrent is incomplete without an evaluation of the incentives to share, as this is their main concern and focus. To evaluate the incentives to share, we set a different limited upload bandwidth to several groups of 50 peers each. Figure 8 shows the latency penalty paid by each group.

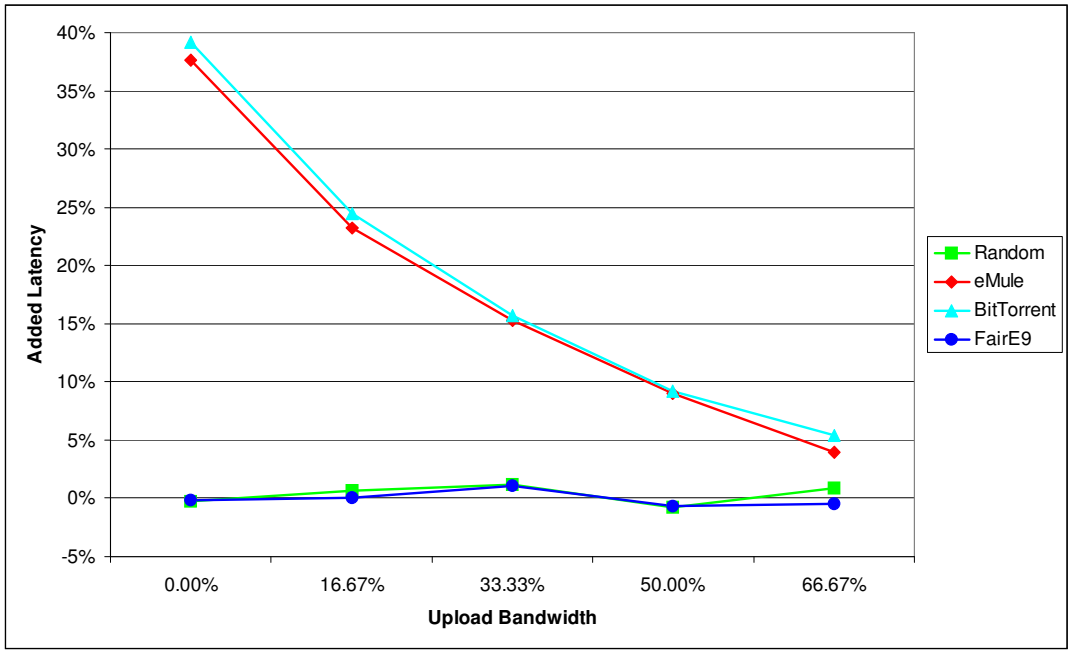


Figure 8 Limited upload bandwidth

It is clear that eMule and BitTorrent impose incentives to upload. But it should be noted that we also measured the number of uploads performed by each group and saw that it correlates with the upload bandwidth presented here. So with these two protocols peers that have only pieces that others do not need, or a limited upload bandwidth, also suffer from long latency. Another important point is that even free-riders in BitTorrent complete their download eventually, paying less than additional 40% in latency. So the incentive to share depends on the trade-off between latency and dedicated upload bandwidth.

4.8. Free-Riders

Even though free-riders are not FairE9's main concern, we find it necessary to compare their effect on FairE9 to their effect on eMule and BitTorrent.

In Figure 9 we compare the effect on the average latency when some of the peers free-ride on the various algorithms. In Figure 10 we focus on the latency gain of the free-riders in comparison to the sharing peers.

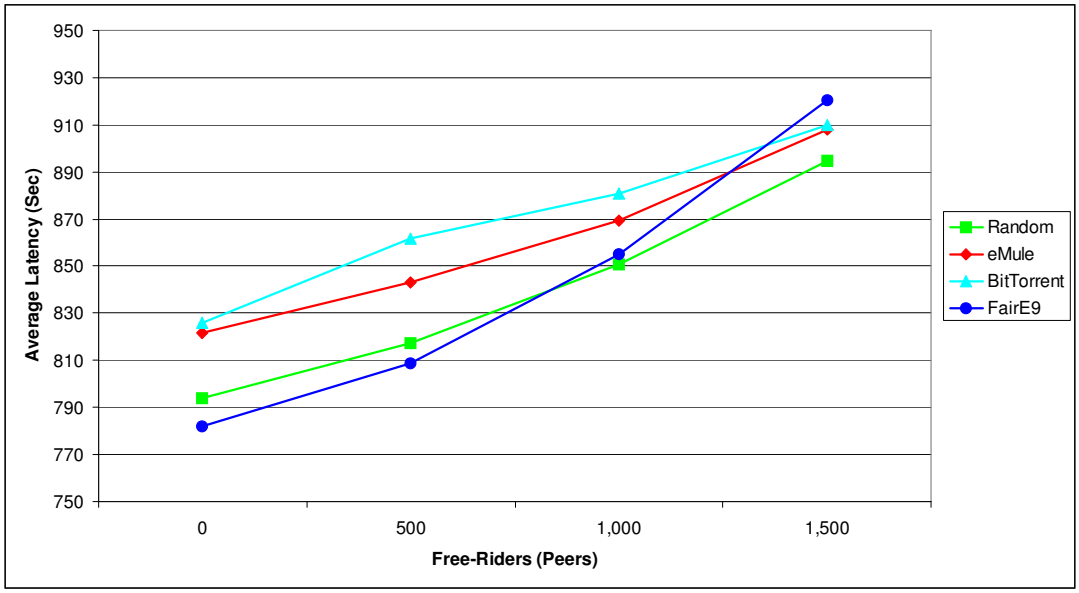


Figure 9 Average latency in the presence of free-riders

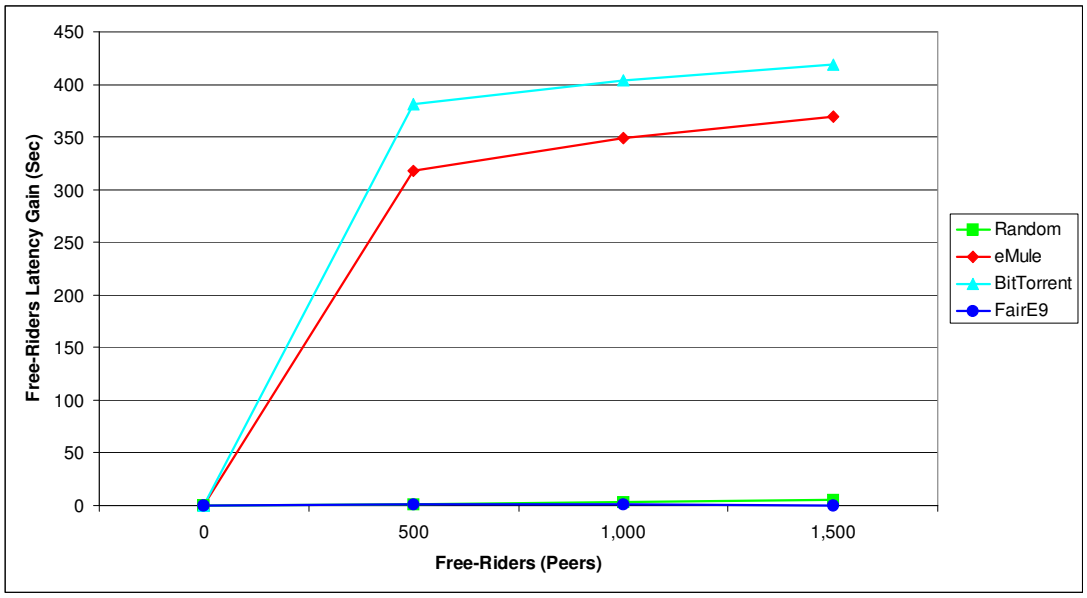


Figure 10 Free-riders compared to sharing peers

FairE9 takes a completely different approach to free-riders than eMule and BitTorrent. Instead of a direct punishment to the free-riding peer, an incentive to cooperate is offered, as cooperative behavior decreases the overall and self latencies.

4.9. Reduce Hard Disk Drive Reads

Sometimes a peer's upload bottleneck is the memory reading speed. This is due to the hard disk drive access time that derives from its head movements. A broadband peer that shares many files made of small pieces, may encounter such a problem. One of the

keys to the solution is to make each peer focus on a small number of distinct pieces to upload. This can not only reduce head movements but it also enables the use of fast cache memory.

Table 5 presents the results of a measured number of distinct pieces uploaded by each peer. Since peers start to share from the moment they have only one piece, even in the random protocols, peers upload less than half of the pieces. Random is a little better than eMule, because the latter sometimes couples peers that exchange distinct pieces on purpose. eMule is a little better than BitTorrent because of a similar reason, as BitTorrent's TFT is a stricter decoupling mechanism.

Protocol	Distinct Pieces Avg
Random	91.83
eMule	93.71
BitTorrent	94.39
FairE9	37.58

Table 5 Distinct uploaded pieces per protocol

FairE9 deals with this phenomenon by having each peer become in practice a major source for only a small number of distinct pieces. More than that, the weight system helps to focus on 1-2 pieces, as illustrated in

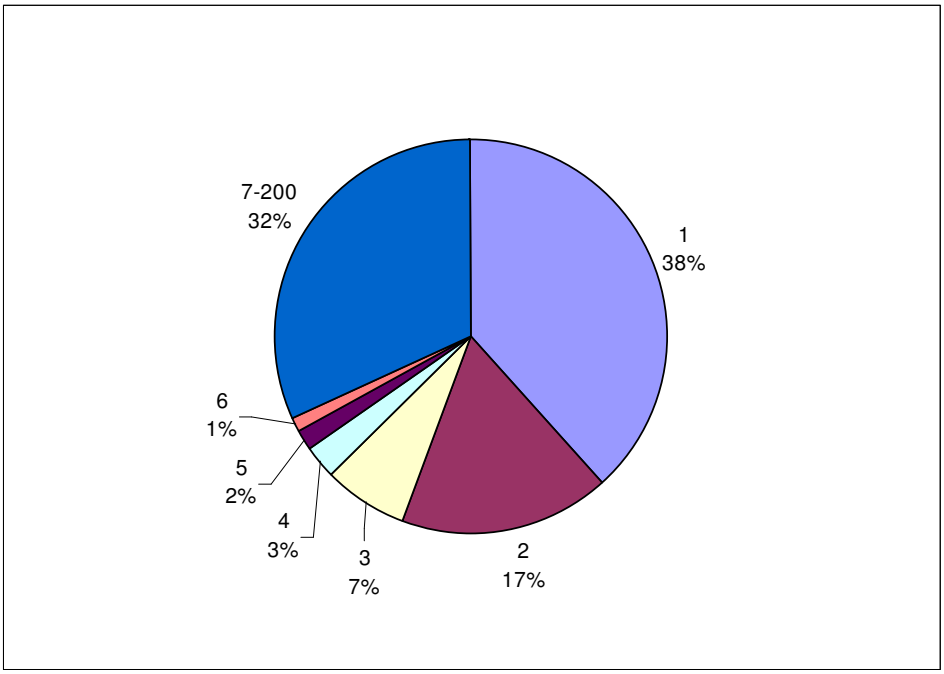


Figure 11. According to it a peer could serve 55% of 1,000 files from 250MB of fast memory.

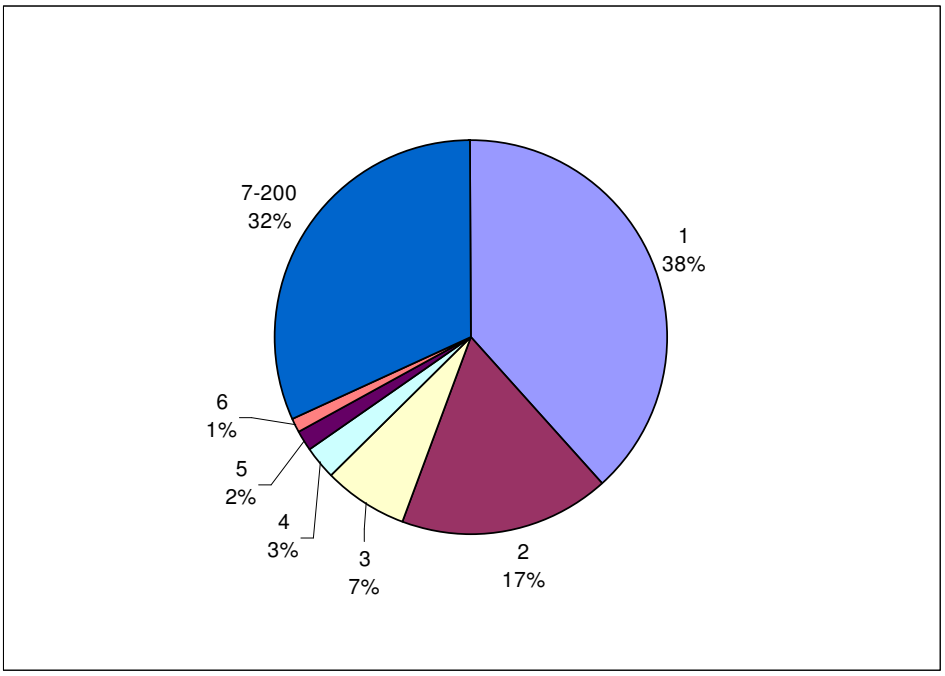


Figure 11 FairE9 uploads by piece position

5. Future Work

FairE9 can be enhanced to be incentives-based and even directly confront free-riders. Since peers are expected to download according to piece-order, the system is ready for direct-reciprocity mechanism. One option is that a peer that wishes to download will be forced to upload simultaneously a lower order piece. This constraint is expected to add latency to the average case, with even more latency to less cooperative peers. Such a scheme does not require the use of long-term memory or the presence of a central authority.

The presented weight formula gives equal importance to the index of the piece on both sides. An unbalanced formula can change the properties of the protocol. We expect that a formula that gives a higher weight to the index on the downloading side will give a boost to newcomers and improve the latency. The trade-off would probably be additional overhead.

We used a balanced permutation function. In some cases it may be more suitable to use an unbalanced function. For example, if a video preview is required it would be better to use a function that tends to put the first file pieces in low indexes. This will distribute the first pieces faster than the other pieces.

6. Conclusions

We presented the design and evaluation of FairE9, a fair system for file-distribution

from a single source, with no central authority. Each peer has an inherent individual piece-order. The combination of the two peers' orders imposes weights on each request. Serving priority is determined by these weights, making the entire process more deterministic and systematic.

As other unstructured P2P networks, it copes well with churn, free-riders, heterogeneous environments and peer failures. In comparison to other unstructured P2P networks, FairE9 also handles flash crowd, balances the upload efforts, welcomes newcomers and reduces the message overhead.

References

[1] J. Jaeyeon, K. Balachander, and R. Michael, "Flash crowds and denial of service attacks: characterization and implications for CDNs and web sites," in *Proceedings of the 11th international conference on World Wide Web* Honolulu, Hawaii, USA: ACM Press, 2002, pp. 293-304.

[2] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh, "Splitstream: High-bandwidth multicast in cooperative environments," in *Proceedings of the 19th ACM symposium on Operating systems principles (SOSP '03)* 2003.

[3] Antony I.T.Rowstron and Peter Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg (Middleware '01),* 2218 ed London, UK: Springer-Verlag, 2001, pp. 329-350.

[4] J.J.D.Mol, D.H.J.Epema, and H.J.Sips, "The Orchard Algorithm: P2P Multicasting without Free-Riding," in *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing (P2P '06)* Washington, DC, USA: IEEE Computer Society, 2006, pp. 275-282.

[5] Vidhyashankar Venkataraman, Paul Francis, and John Calandrino, "Chunkyspread: Multi-tree Unstructured Peer-to-Peer Multicast," in *Procedding of the 5th International Workshop on Peer-to-Peer Systems (IPTPS '06)* 2006.

[6] Anwitaman Datta, Ion Stoica, and Mike Franklin, "LagOver: Latency Gradated Overlays," in *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS '07)* 2007.

[7] Vidhyashankar Venkataraman, Paul Francis, and John Calandrino, "Climber: An Incentive-based Resilient Peer-to-Peer," in *Proceedings of the Seventh International Workshop on Peer-to-Peer Systems (IPTPS 2008)* 2008.

[8] Tsuen-Wan "Johnny" Ngan, Dan S.Wallach, and Peter Druschel, "Incentives-Compatible Peer-to-Peer Multicast," in *Proceeding of the 2nd Workshop on Economics of Peer-to-Peer Systems* 2004.

[9] BitTorrent, "http://bittorrent.com," 2004.

[10] Gnutella, "http://www.gnutella.com," 2000.

[11] eMule, "http://www.emule-project.net," 2002.

[12] Nazareno Andrade, Miranda Mowbray, Aliandro Lima, Gustavo Wagner, and Matei Ripeanu, "Influences on cooperation in BitTorrent communities," in *Proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems* ACM Press New York, NY, USA, 2005, pp. 111-115.

[13] Michal Feldman and John Chuang, "Overcoming free-riding behavior in peer-to-peer systems," *ACM SIGecom Exchanges*, vol. 5, no. 4, pp. 41-50, July2005.

[14] Robert Axelrod and William D.Hamilton, "The Evolution of Cooperation," *Science*, no. 211, pp. 1390-1396, 1981.

[15] Ashwin R.Bharambe, Cormac Herley, and Venkata N.Padmanabhan, "Analyzing and improving BitTorrent performance," Microsoft Research,MSR-TR-2005-03, Feb.2005.

[16] Gang Wu and Tzi-cker Chiueh, "How Efficient is BitTorrent?," in *Proceeding of Multimedia Computing and Networking (MMCN '06),* 6071 ed 2006.

[17] Seung Jun and Mustaque Ahamad, "Incentives in BitTorrent induce free riding," in *Proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems (P2PECON '05)* New York, NY, USA: ACM Press, 2005, pp. 116-121.

[18] Wendy Myrvold and Frank Ruskey, "Ranking and Unranking Permutations in Linear Time," *Information Processing Letters*, vol. 79, no. 6, pp. 281-284, 2001.

[19] Martin Mares and Milan Straka, "Linear-Time Ranking of Permutations," in *Algorithms – ESA 2007* Springer Berlin / Heidelberg, 2007, pp. 187-193.

[20] Bram Cohen, "Incentives Build Robustness in BitTorrent," in *Proceeding of the 1st Workshop on Economics of Peer-to-Peer Systems (IPTPS '03)* 2003.

Appendix A – Simulator Source Code

תקציר

רשתות P2P נמצאות באור הזרקורים תודות לתפוצה הרחבה של רשתות שיתוף-קבצים. אלגוריתמים רבים להפצת-קבצים הוצעו ומומשו. על הפתרונות השונים להתמודד עם סביבה הטרוגנית ולא יציבה, בה צמתים עשויים להצטרף ולנטוש בתדירות גבוהה (churn). לעתים לא ניתן להניח שיתקיים שיתוף פעולה מלא בין הצמתים. נושאים אלו הופכים את גישת המערכות המובנות לפחות מעשיות. אפילו כמה מהאלגוריתמים הנחשבים בלתי-מובנים משמרים יחסי אב-בן לטווח-ארוך. אלגוריתמים בלתי-מובנים קיימים להפצת-קבצים פועלים לרוב היטב ברשת האינטרנט. אך כמה מהצמתים המשתתפים עשויים לסבול מאתחול איטי או תקורה גבוהה בגלל האקראיות של בחירת הצמתים והפיסות להעלאה והורדה.

במסמך זה אנו מציעים מערכת הוגנת בלתי-מובנית להפצת-קבצים ממקור יחיד, ללא סמכות מרכזית. הפרוטוקול המוצע הוגן בהיבטים של חלוקת העומס וזמן-סיום בכל צומת. הוא מבוסס על אלגוריתם-משקלים מקורי המסייע לצמתים לקבוע איזו פיסה לבקש מאיזה צומת, באופן המגדיל את סיכוייהם לקבל שירות. כך גם מושגת הקטנה של התקורה. האלגוריתם המוצע מקדם מצטרפים-חדשים תוך כדי גילוי עמידות בפני תחלופה גבוהה, עמידות בפני אוכלי-חינם, והסתגלות לסביבה בעלת רוחב פס מגוון.

תוכן העניינים

האוניברסיטה הפתוחה
המחלקה למתמטיקה ולמדעי המחשב

FairE9: הפצת תכנים הוגנת על-גבי P2P בלתי מובנה

עבודת תזה זו הוגשה כחלק מהדרישות לקבלת תואר
"מוסמך למדעים" .M.Sc במדעי המחשב
באוניברסיטה הפתוחה
החטיבה למדעי המחשב

על-ידי
איל זהר

העבודה הוכנה בהדרכתה של ד"ר ענת לרנר

ינואר 2009