A Further Look at the Distance-Availability Weighted Piece Selection Method

A BitTorrent Piece Selection Method for On-Demand Media Streaming

Petter Sandvik and Mats Neovius
Department of Information Technologies
Åbo Akademi University
and
Turku Centre for Computer Science
Turku, Finland
e-mail: {petter.sandvik, mats.neovius}@abo.fi

Abstract—**During the last few years, BitTorrent has become a popular way of transferring large files over the Internet. However, the original out-of-order nature of the BitTorrent protocol has made it difficult to enable playback of media files that have not yet been fully transferred. In this paper we describe a piece selection method which we believe will enable simultaneous playback of the transferred media file without impacting on the speed and quality of the transfer. The distance-availability weighted method compromises between selecting rare pieces and pieces which are soon to be played back, making playback possible before the transfer is complete. In our simulations, we have compared our piece selection method with other proposals for on-demand streaming media using a BitTorrent-like setup, with our method giving similar or better results.**

Keywords-media, on-demand, peer-to-peer, streaming, BitTorrent, simulation

I. INTRODUCTION

In [1], we presented the distance-availability weighted method; a BitTorrent piece selection method for on-demand streaming. BitTorrent is originally a peer-to-peer file sharing protocol and an application, designed by Bram Cohen and first released in July 2001 [2]. During the following years BitTorrent evolved into one of the most popular peer-to-peer protocols [3][4]. Unlike earlier popular peer-to-peer file sharing applications such as Napster and Kazaa, the use of BitTorrent usually starts by clicking a link in a web browser, and Cohen suggests that "ease of use has contributed greatly to BitTorrent's adoption, and may even be more important than [...] the performance and cost redistribution features" [5]. Another major difference between the earlier peer-to-peer file sharing applications and BitTorrent is the lack of central server in the latter, in the sense that BitTorrent users are not all connected to each other through one server.

While BitTorrent is popular, there are also other reasons for choosing it as the basis for a peer-to-peer based media streaming system such as the one we have in mind. Several applications using the protocol, as well as the protocol itself, are open, which makes understanding and modifying more easily possible than if the staring point was a proprietary product. For our purposes an even more important aspect is that all the logic involved in the file transfer is contained on the client side, which makes it possible, at least in theory, to have an on-demand content streaming BitTorrent client participate in content transfer with regular, non-streaming, BitTorrent clients.

While the original BitTorrent protocol was not designed for streaming, it has already been argued [6] that with some modifications, it would be possible to create a streaming media solution based on BitTorrent. Indeed, there are proprietary and commercial efforts to create exactly such a thing, for instance "to turn BitTorrent into a point-click-watch experience much more similar to YouTube" [7], and in [1] we described our proposal for a solution that would enable file sharing in such a way that content could be played back while downloading. In this paper we will further describe our proposal and show that it is a modification to the BitTorrent protocol, which would allow it to function as an on-demand media streaming solution.

The rest of this paper is organised as follows: in Section II, we describe BitTorrent and its terminology, as used throughout this paper. Section III consists of the requirements of an on-demand streaming media application in general, and what assumptions we will make for a BitTorrent streaming application to be feasible. In Section IV we present our proposed piece selection method, and in Section V, we compare it to existing piece selection methods that could be used for on-demand streaming. In Section VI, we list a selection of related works. The paper is concluded in section VII with a discussion about our findings and possible future work on this subject.

II. BITTORRENT

Since BitTorrent was introduced by Bram Cohen in 2001 it has evolved. While the terminology used in [5] could be seen as a standard, the terminology used in this paper will be based on that of [8]. This terminology is close to that of the application Vuze, formerly known as Azureus, a BitTorrent client often used as a basis for research applications [9][10][11].

A. BitTorrent Terminology

- **Torrent**. A torrent consists of a single file, or a collection of files, to be shared, and the associated metadata. The metadata, and therefore the torrent itself, is uniquely identified by its info-hash [12].

- **Tracker**. A tracker is a piece of software that is involved in keeping track of which peers are involved in the transfer of a particular torrent, using the info-hash of the torrent. Each torrent can be associated with many trackers. Additions to the BitTorrent protocol have enabled peer discovery through other means, such as distributed hash tables and peer exchange, and thus the use of trackers is no longer required. Whether a tracker is used or not does not affect the file transfer, and this is of little importance to us.
- **Pieces** and **blocks**. The data of a torrent is divided into pieces, and each piece is divided into blocks, also called sub-pieces. A piece is typically 256 kilobytes in size [8][13] while a block is typically 16 kilobytes in size [5][8]. A piece must be complete, that is, all blocks of it must have been downloaded, before the piece can be transferred to another peer.
- **Torrent index file**. Also known as a ".torrent" [5], a torrent index file contains information about the torrent, such as the universal resource locator (URL) of the tracker (or trackers, if any), piece size of the torrent, names and sizes of the files in the torrent, as well as SHA-1 hashes of all the pieces. This file is generally hosted on a web server and downloading of a torrent starts by opening the file with a BitTorrent application.
- **Interested** and **to choke**. If peer B has pieces, which peer A does not have, peer A is interested in peer B, otherwise peer A is uninterested in peer B. If peer B decides not to send data to peer A, peer B chokes peer A, and if peer B decides to send data to peer A, peer B unchokes peer A.
- **Peer set** and **active peer set**. A peer set consists of all the other peers one peer is connected to, and the active peer set consists of those peers it is currently sending data to, i.e., its unchoked peers.
- **Seed**. A peer that has all pieces of a torrent and therefore only sends data is called a seed.
- **Availability**. We define availability of a piece as the number of peers in the peer set who have that specific piece.

B. How BitTorrent Data Transfer Works

When transferring data, BitTorrent needs to decide what data to request (piece selection) and which peers to choke or unchoke (peer selection). The reference BitTorrent implementation begins by selecting pieces to download at random, until one complete piece has been downloaded. BitTorrent then switches to a rarest-first piece selection method. The rarest-first selection method selects the piece that the fewest peers have, i.e., the piece with the lowest availability, as the first piece to request. This has the effect of reducing the possibility that one piece may become unavailable, as a lower number of peers having a piece makes other peers more likely to request that particular piece, thereby increasing the number of peers that will have that piece. When a single block from a piece has been downloaded, other blocks from that piece are given highest priority, in order to have as few incomplete pieces transferred as possible. BitTorrent then keeps selecting the rarest pieces first, until all remaining blocks in all remaining pieces have been requested, at which time all remaining blocks are requested from all peers in the active peer set. This is done so that one slow peer cannot prevent the whole download from completing.

To create an incentive for peers to upload as well as download, the reference BitTorrent implementation uses a tit-for-tat (TFT) peer selection strategy. Every ten seconds, which peers to unchoke is evaluated based on the rate of data sent, with the fastest peers chosen as the peers to unchoke. Additionally, there is also one interested peer unchoked at random, re-evaluated every thirty seconds. This randomly chosen peer is called the optimistic unchoke [5][8] and exists for two reasons: to allow new peers a chance to enter the TFT game, and to potentially discover faster peers that could become regular, non-optimistic, unchokes [8][9]. This approach is not necessarily the optimal way of peer selection, and alternative approaches have been suggested, such as [9]. However, the basic TFT strategy remains an essential part of the BitTorrent protocol as used today.

After a download is finished, the peer may continue to participate in sending data to other peers, and in many cases the peer is actually encouraged to do so. In this case choosing peers based on how much they send is of course not possible, and thus the reference BitTorrent implementation then switches to sending to the peers that can receive data the fastest [5]. However, this feature could be exploited, and later clients have switched to choosing peers to send to randomly [9].

III. REQUIREMENTS FOR STREAMING

We define streaming as the transport of data in a continuous flow, in which the data can be used before it has been received in its entirety. In this context, on-demand streaming is essentially playback, as a stream, of pre-recorded content, at the request of a user. This is in contrast to live streaming, which is playback, as a stream, of content, which is not pre-recorded but rather created and transmitted practically simultaneously. Concerning an on-demand peer-to-peer media streaming system, we make the following initial observations:

Each peer must have a download bandwidth at least as large as the playback bit rate of the media. Unlike a peer-to-peer file sharing system, the media is played while being received, and therefore cannot be received slower than it should be played back. Furthermore, if the media is to be received faster than real-time, it must also be sent faster than real-time. Therefore, the average upload bandwidth of all peers must be larger than the playback bit rate of the media, although an individual peer may have an upload bandwidth smaller than that.

We will make the following assumptions regarding a BitTorrent-based on-demand media streaming system: The torrent will consist of only one complete media file, unlike in file sharing where several files can be combined in one torrent. Additionally, in a file sharing system the time an

individual peer participates is difficult to estimate and not dependent on the content received. In an on-demand peer-to-peer streaming system it can be estimated more easily, and although peers may of course leave the system at any time, our assumption is that a typical peer will enter the system when starting playback and leave the system some time after playback is complete, which means some time after all pieces of the content have been received. We will also assume that for each piece there will always be at least one peer holding it, that is, we assume that we do not encounter the situation where the availability of any piece is zero, because that would lead to a situation where complete playback is not possible.

Internet connections are typically symmetric, with the same bandwidth available for sending and receiving data, or asymmetric with a higher download bandwidth than upload bandwidth. Therefore, a typical peer will be able to receive data at least as fast as it can send data. Additionally, all pieces must be requested, resulting in a complete file once the transfer is complete, unless the peer leaves before finishing playback. Furthermore, each peer will keep and make available all the pieces it has received, for as long as the peer is participating. Each peer must therefore have enough space to store the entire media file.

Ideally, the piece selection algorithm in our BitTorrent-based streaming system should comprise the following behaviours: When the ratio of seeds to peers is high, our piece selection method should prefer pieces close to being played back rather than rare pieces, because with a large number of complete sources there is no need for downloading rare pieces to ensure the future availability of all pieces. In the extreme case, where our client has the only incomplete copy of the content, there is no obvious downside to requesting pieces sequentially. On the other hand, when the ratio of seeds to other peers is low, our piece selection method should also choose rare pieces to improve the overall availability of the pieces, while still requesting enough pieces in sequence to make continuous playback possible.

Although we specified earlier that peers should be able to download faster than the playback rate of the media, there will be variations in how much faster the peers will be able to receive data. Ideally, our piece selection method will be able to adapt to these kinds of differing conditions. For instance, if a peer can download data only slightly faster than the playback rate, our piece selection method should ensure that the peer requests data mostly sequentially, while a peer that can download the media much faster than its playback speed should also frequently request rare pieces in order to improve the overall availability of the pieces. With that in mind, we present our proposal for a piece selection method for on-demand streaming.

IV. THE DISTANCE-AVAILABILITY WEIGHTED METHOD

The idea behind the distance-availability weighted method for piece selection (DAW) is to strike a balance between lots of consecutive pieces, which is good for playback, and requesting rarest pieces first, which is good for piece availability. In other words, we want to balance requesting between distance (in the sequence of pieces) and availability.

We start by having a small, fixed buffer of size k. The priority for requesting the pieces in the buffer will be the highest, here represented as 1. Outside the buffer we will calculate the priority for each not yet requested piece as

$$\text{Priority} = 1/((P_r - P_c) * m_r)$$

where P_r is the sequence number of a particular piece, m_r is the number of peers who hold that piece, and P_c is the sequence number of the current last piece in the buffer. In other words, $P_r - P_c$ is the distance to a particular piece and m_r is the availability of that piece. The priority for a piece outside the buffer is therefore never more than 1, with a priority of 1 occurring only in the rare situation that the piece immediately outside the buffer is held by exactly one peer. Pieces that are further from being played back or held by more peers are given lower priorities, while a short distance from the buffer or a low availability increases the priority.

The availability of a particular piece and its distance from the last piece in the buffer are here equally weighted when determining the priority. However, we do not claim that giving equal weights to distance and availability is in any way the optimal solution. We have not extensively tested different weights, but it seems likely that factors such as the total number of pieces, the number of peers, and the network speed of the peers, will have an effect on how the distance and availability should be weighted for optimal performance. As it would not be possible to test all different combinations of weights under all circumstances, we choose here to weigh them equally for the sake of simplicity.

It should be noted that when we talk about availability of a piece we do not mean how many peers in total are involved in the torrent and hold that particular piece. What we actually look at is how many peers in one peer's active peer set hold that particular piece. As one peer need not necessarily be connected to all other peers, especially if the total number of peers is very large, we must by necessity look at the system from one peer's point of view. Another point worth mentioning is that the above priority calculation holds even if we allow playback to start somewhere else than from the beginning. Not yet requested pieces earlier in sequence than the last piece of our playback buffer will get negative priorities, and therefore will not be requested until all pieces higher in sequence have been requested.

V. COMPARISON WITH OTHER PIECE SELECTION METHODS

We have done two separate sets of simulations. The first set, the results of which also appeared in [1], is less exhaustive and compares our DAW piece selection method to two others:

- **Sequential Method**. This represents how a straightforward streaming would work, by always requesting pieces in the same order as they appear in the torrent.

- **Rarest-First Method with Buffer (RFB)**. This is the original BitTorrent piece selection method, slightly modified to better support streaming media. This is done as follows: we add a small buffer of fixed size, so that *k* pieces after the currently playing one are requested with the highest priority. If all *k* buffer pieces have been requested, we use the rarest-first method on the remaining part of the media. Another small modification is that if more than one piece have the same availability the original rarest-first method chooses between them randomly [12], while we always choose the one closest to being played back, as in the rarest-first method mentioned in [6].

In our second set of simulations, we add another piece selection method to the comparison:

- **BitTorrent Streaming (BiToS)**. This piece selection method is described in [6]. Pieces not yet downloaded are divided between a small high-priority set, with pieces close to being played back, and a larger remaining pieces set, with lower priority. The probability of choosing a piece from the high-priority set is *p* and the probability of choosing a piece from the remaining pieces set is similarly 1-*p*. Within each set, pieces are chosen rarest-first, with the aforementioned modification that if there is more than one piece with equal availability, the one closest to being played back is chosen. In [6], *p* was chosen to be 0.8 and we have used the same number here.

A. Our First Set of Simulations

In these simulations, we have focused on the piece selection algorithms, and the results therefore do not reflect real-world performance of applications. Our main focus has been on two things: the percent of requests going to the original source over time, and the availability of the last piece (which is also the rarest piece, in these cases) over logical time. The first one of these we want to be as low as possible, because a good peer-to-peer system should distribute the load equally over as many peers as possible and therefore not have a proportionally high amount of requests directed to the original source. The second one we want to be high, as we want the availability of the rarest piece to increase, as that signifies redundancy and robustness of the system. In these simulations we also assume that peers have the ability to send data to as many other peers as necessary, effectively creating a situation where a peer will always get the piece it requests.

The number of pieces was chosen to be 1000; large enough to study the behaviour of different piece selection methods over long periods of time. The buffer size for both DAW and RFB was set to 8 pieces; large enough for smooth playback but small enough that filling the buffer should not impact n overall performance. All simulations were done up to 800 logical time units; at the rate of one request per time unit we therefore never reach the situation that any peers have finished downloading, instead focusing on what happens from the start onwards.

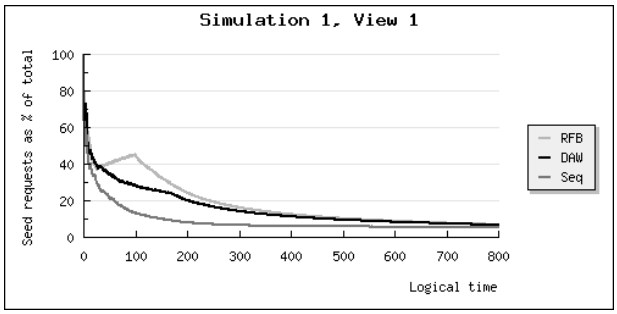

Figure 1. Seed requests as percentage of total requests over logical time, with peers joining at regular intervals.

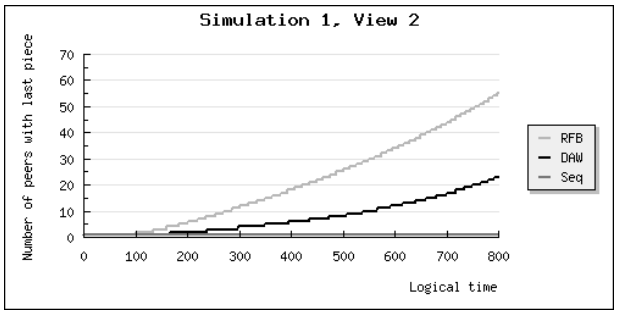

Figure 2. Availability of last piece over logical time, with peers joining at regular intervals.

In the first simulation, to stress the system we chose to have one seed and 100 regular peers, the latter arriving at regular intervals (one new peer every two logical time units). Fig. 1 shows that DAW here results in a lesser burden for the seed than RFB, although it is not as good as the sequential method. However, as Fig. 2 shows, the DAW does not increase the availability of the last piece as much as RFB. With RFB many rare pieces are requested early. These pieces are only available from the seed, and as seen in Fig. 1 the burden on the seed drops only after 100 logical time units. This corresponds to the point where there is no piece left where the seed is the only source, as can be seen in Fig. 2. The same drop can be seen, less dramatically, for DAW around logical time 170, with the same explanation.

The second simulation is identical to the first one, except that all regular peers join simultaneously. As can be seen in Fig. 3, the sequential method does not work in this theoretical situation, while DAW is the better suited one of the other two. Fig. 4 shows the situation as very similar to the one in Fig. 2, except that with more peers from the start it takes less time to increase the availability of the last (rarest) piece with DAW and RFB.

For our third simulation, we chose a similar setup to the first one, but with ten seeds instead of one. Fig. 5 shows the average percentage of requests over logical time to each one of the ten seeds, and DAW is again a less taxing choice than RFB. A possible reason for this can be seen in Fig. 6;

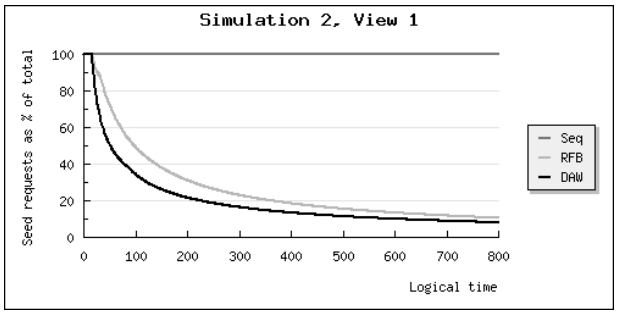

Figure 3. Seed requests as percentage of total requests over logical time, with peers joining simultaneously.

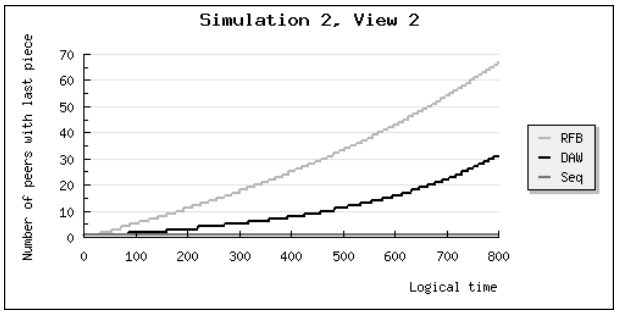

Figure 4. Availability of last piece over logical time, with peers joining simultaneously.

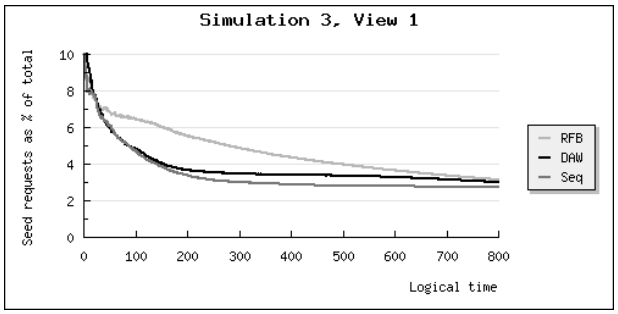

Figure 5. Average requests to a seed as percentage of total requests over logical time, with peers joining at regular intervals.

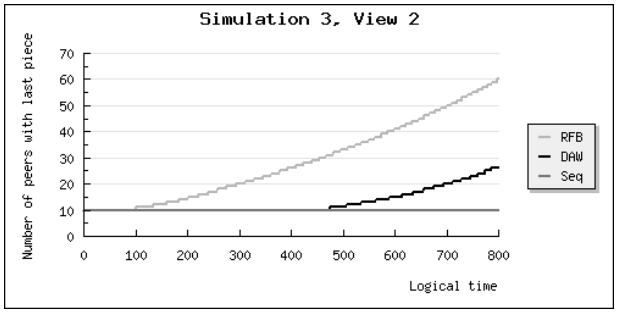

Figure 6. Availability of last piece over logical time, with peers joining at regular intervals.

compared to DAW, RFB spends a lot of time requesting rare pieces although there is no immediate reason for doing so.

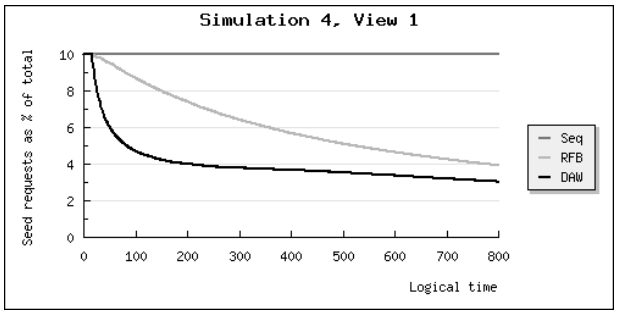

Figure 7. Average requests to a seed as percentage of total requests over logical time, with peers joining simultaneously.

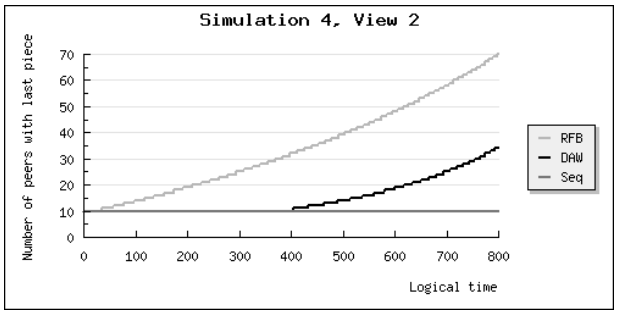

Figure 8. Availability of last piece over logical time, with peers joining simultaneously.

Our fourth simulation is a combination of the second and third ones, with ten seeds and all the regular peers starting at the same time. In Fig. 7 we see that the DAW piece selection method is again less demanding on the seeds than the RFB method, with the sequential method making on average a constant ten percent of the requests to each seed in this theoretical case. Fig. 8 is very much like Fig. 6, showing that the distance-availability weighted method does not spend a lot of time requesting the rarest piece until fairly late.

B. Our Second Set of Simulations

Compared to our first set of simulations, our second set is a step or two closer to reality. Unlike our first set of simulations, where a peer would always get the piece it requested, we have here introduced limits to how many other peers the seeds and regular peers can send to. This introduces the possibility that a peer will not have the piece that it is supposed to be playing back. In our simulations we have dealt with that situation in three different ways:

- **Skip**. We ignore the piece we should be playing back and just note that that piece has been skipped. This is the method favoured by BiToS [6], and it should be noted that for this to work in practise the format of the media content must be tolerant of missing data.
- **Stop**. If we are missing the piece that should be played back, we stop playback until we have been able to receive all pieces in our buffer (or high-priority set), after which we resume playback. From an end user point of view this is similar to how many current on-demand streaming media systems work,

in that not receiving data fast enough means that playback is paused until an amount of consecutive data has been received.

- **Skip and stop**. If we are missing the piece we are supposed to play back we skip it, but if our buffer (or high-priority set) is completely empty we stop playback until we have received all pieces in it. This should in theory generate less complete stops than by always stopping, and possibly also less skips than by just skipping.

What we actually measure in these simulations is three things: firstly, the playback position of the peers after a certain time; secondly, the number of skips and/or stops encountered before that time; and thirdly, the percentage of pieces that should have been transferred that were actually skipped and/or the number of stops per 100 pieces played back, respectively. What we are looking for is thereby a high number for the playback position, but low numbers for the amount of skips and stops.

In all these simulations, we have 10 seeds and 90 regular peers. The seeds can upload to 8 peers simultaneously and the regular peers to 2 peers simultaneously, at the rate of half the playback speed for each peer. The regular peers request new pieces twice as fast as the playback speed. In all the following figures we look at the situation after a logical time of 800. In simulations 5, 6 and 7 the regular peers join simultaneously, while in simulations 8, 9 and 10 they join at regular intervals; similarly to simulations 1 and 3. All simulations were run multiple times and the maximum reported here is the maximum for any peer during any run, the minimum reported is the minimum for any one peer duing any run, and the average reported is the average for all peers over all runs.

Our simulation number five concerns the situation where any pieces not transferred are skipped completely. Fig. 9 shows the maximum, average and minimum playback positions of peers using the BiToS, DAW, RFB and sequential piece selection methods. BiToS seems worse than the others in this case, but it must be pointed out that in BiToS we always spend about 20% of our time downloading pieces not close to being played back, and therefore it is reasonable to expect it to take longer for playback to start. In a best-case scenario this would lead to a lower number of pieces skipped later on, but as we see in Fig. 10, there is not a significant difference in the absolute number of skipped pieces, and if we look at the relative numbers, i.e., the percentage of pieces that should have been played back that were skipped, as in Fig. 11, we find that arguably BiToS is worse than the other three, although the differences are small.

Our simulation number six concerns the situation where the piece to be played back not being available leads to a complete stop of the playback. Fig. 12 shows that the distance-availability weighted method here leads to the furthest playback position. In Fig. 13 we see that despite the good results for the playback position, DAW also does pretty well in the absolute number of stops by coming in second. Fig. 14 shows the relative number of stops, i.e., the number of stops per 100 pieces played back, and there is not a big difference between the four piece selection methods.

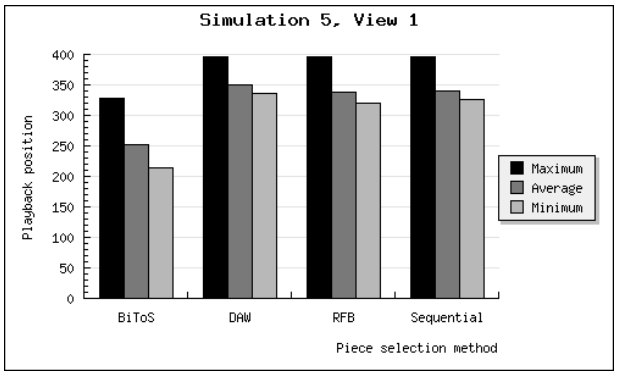

Figure 9. Playback positions of peers when playback stops for missing pieces, with peers joining simultaneously.

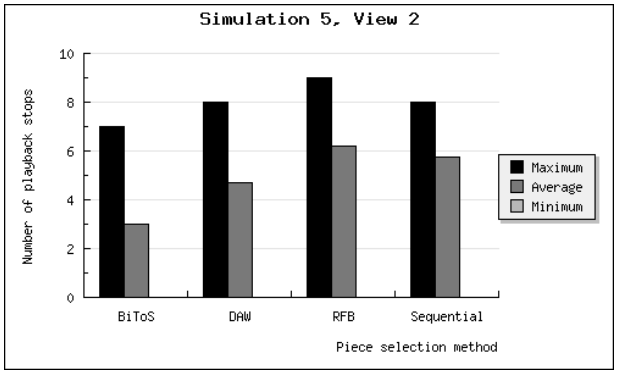

Figure 10. Number of playback stops for each peer when playback stops for missing pieces, with peers joining simultaneously.

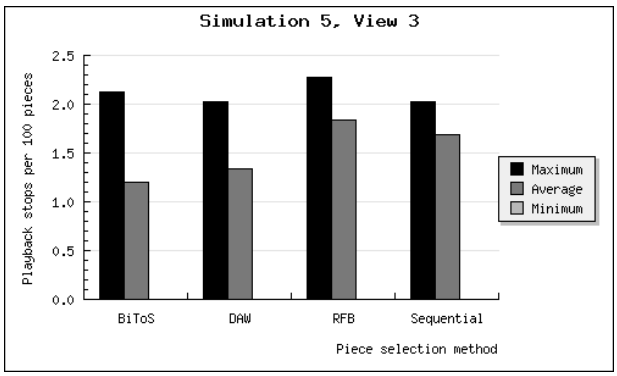

Figure 11. Playback stops per 100 pieces when playback stops for missing pieces, with peers joining simultaneously.

Simulation number seven seems to be the one with the most diverse results so far. Here we have the situation that if the piece to be played back is not received, we skip it, but if we do not have any of the pieces in our buffer or high-priority set, we stop. Fig. 15 shows the playback positions of the peers, and the situation is not very different from in the two preceding simulations, with DAW slightly ahead of the others and BiToS slightly behind. However, in Fig. 16 we notice that BiToS seems to skip a lot more than the others, and in Fig. 17 we notice that BiToS stops a lot less often.

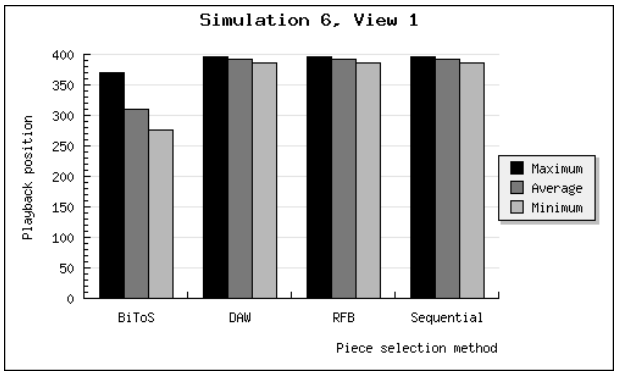

Figure 12. Playback positions of peers when playback skips missing pieces, with peers joining simultaneously.

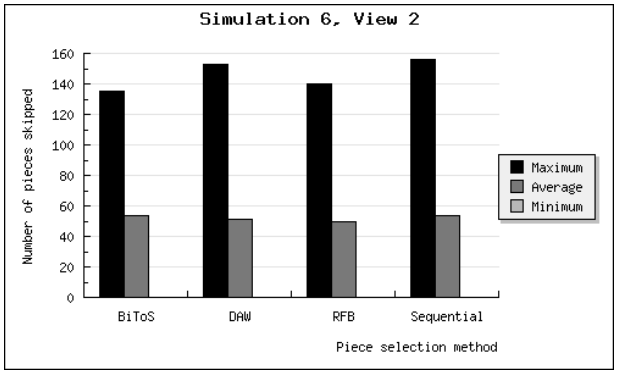

Figure 13. Number of pieces skipped for each peer when playback skips missing pieces, with peers joining simultaneously.

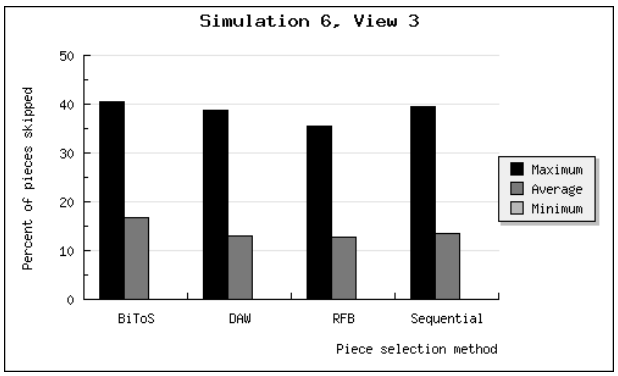

Figure 14. Percent of pieces skipped for each peer when playback skips missing pieces, with peers joining simultaneously.

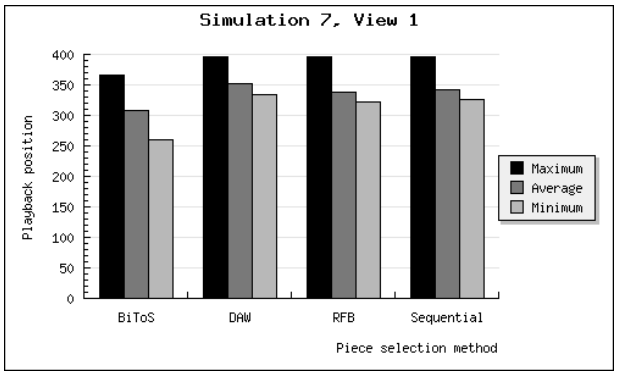

Figure 15. Playback positions of peers when playback skips and stops, with peers joining simultaneously.

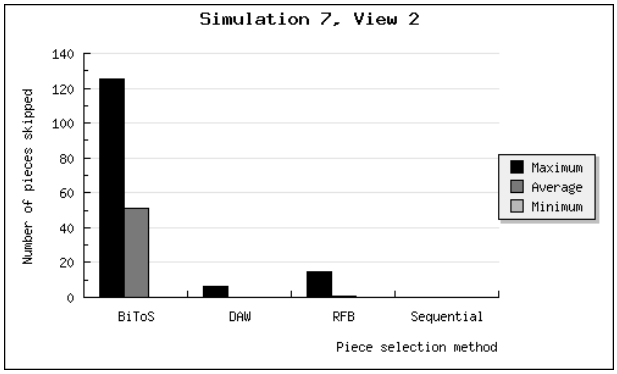

Figure 16. Number of pieces skipped for each peer when playback skips and stops, with peers joining simultaneously.

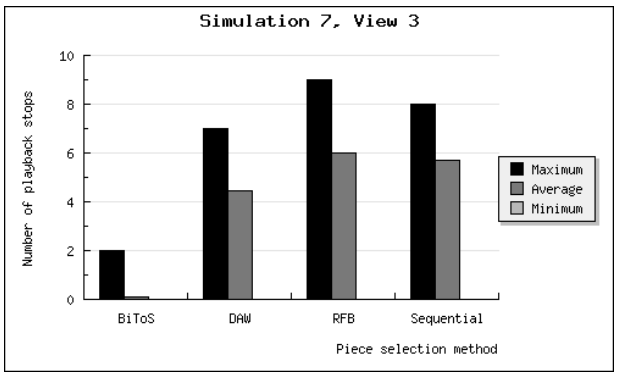

Figure 17. Number of playback stops for each peer when playback skips and stops, with peers joining simultaneously.

This can be explained by how these piece selection methods work. BiToS always requests pieces in a rarest-first, out-of-order fashion, and therefore it is likely that even if the piece to be played back is missing, the high-priority set is not empty, and therefore we just skip. DAW and RFB as described in this paper both have a small buffer in which pieces are requested in-order, and therefore it is likely that if the piece to be played back is missing, the whole buffer is empty, and therefore we stop. In the case of the sequential method, we always request sequentially, and therefore if the piece to be played back is missing, the following k pieces are also missing and we stop.

The following three simulations are similar to the previous ones, except that the regular peers do not join simultaneously. Figures 18, 19 and 20 show the result of simulation 8, where playback stops if the piece to be played back is not available, and the regular peers join at intervals. Because peers do not join at once, the difference between the maximum playback position and the minimum playback position is larger than in simulation 5 (compare Figures 9 and 18). Fig. 19 shows the absolute number of playback stops and Fig. 20 the number of playback stops per 100 pieces, and we see that the DAW and sequential piece selection methods are almost equal in this case, with both

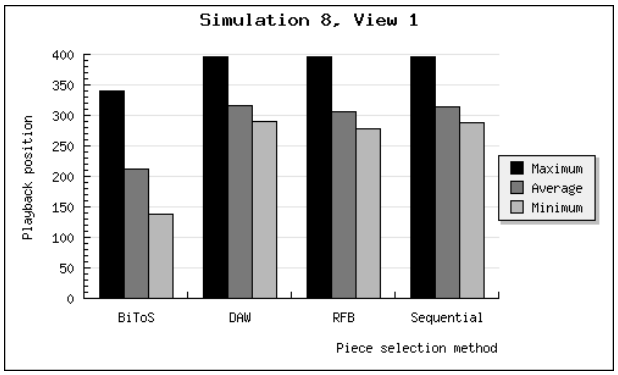

Figure 18. Playback positions of peers when playback stops for missing pieces, with peers joining at regular intervals.

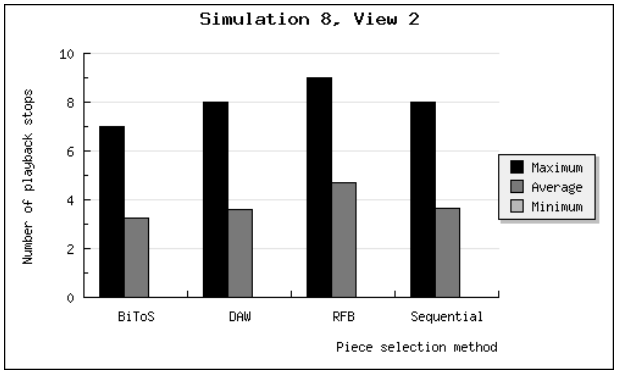

Figure 19. Number of playback stops for each peer when playback stops for missing pieces, with peers joining at regular intervals.

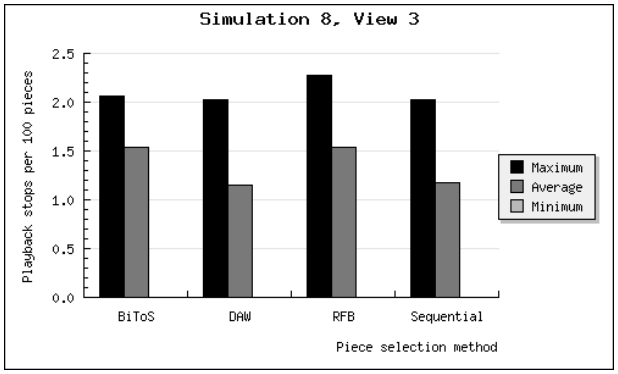

Figure 20. Playback stops per 100 pieces when playback stops for missing pieces, with peers joining at regular intervals.

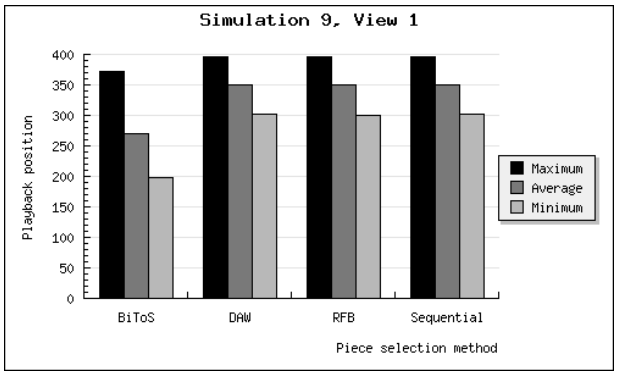

Figure 21. Playback positions of peers when playback skips missing pieces, with peers joining at regular intervals.

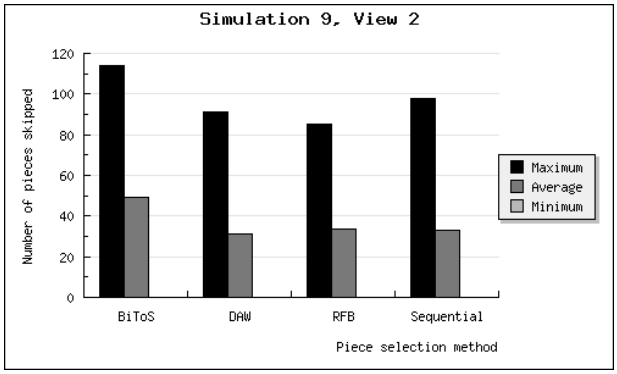

Figure 22. Number of pieces skipped for each peer when playback skips missing pieces, with peers joining at regular intervals.

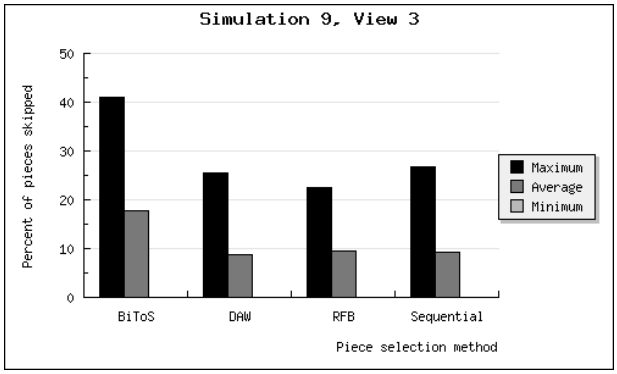

Figure 23. Percent of pieces skipped for each peer when playback skips missing pieces, with peers joining at regular intervals.

being better than the RFB method and in the relative case also better on average than BiToS.

Fig. 21 is comparable to Fig. 12, and again the larger difference between the maximum and minimum playback positions is caused by peers joining at intervals, instead of simultaneously. Comparing Fig. 22 and Fig. 13 we notice that not having all peers joining at once improves the result for the piece selection method utilising some form of sequential requests, leaving BiToS behind. This is emphasised in Fig. 23 where we see the percent of skipped pieces instead of the absolute amount.

Simulation 10 is comparable to simulation 7, and comparing Fig. 24 with Fig. 15 shows that also in this case the peers joining at intervals has the effect of putting BiToS further behind when it comes to playback position. Fig. 25 and Fig. 26 show that as in simulation 7, the nature of BiToS makes it skip more often than stop, while the sequential buffer used in the DAW and RFB piece selection methods make them behave more like the sequential method.

So far, we have only compared the piece selection methods to each other but not discussed the actual figures. While this is of course a very theoretical simulation, the bandwidth figures we have used suggest that playback

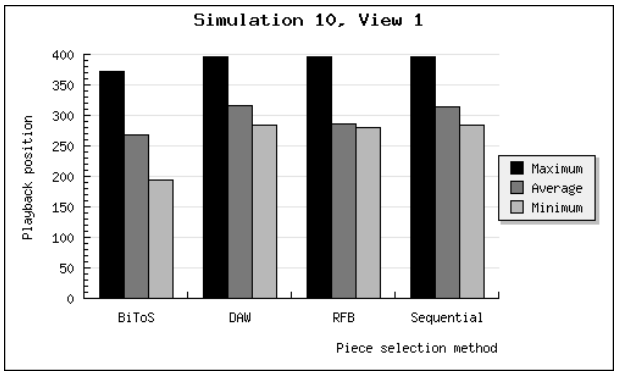

Figure 24. Playback positions of peers when playback skips and stops, with peers joining at regular intervals.

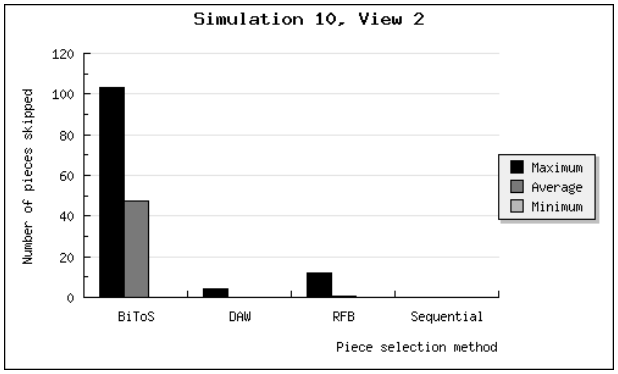

Figure 25. Number of pieces skipped for each peer when playback skips and stops, with peers joining at regular intervals.

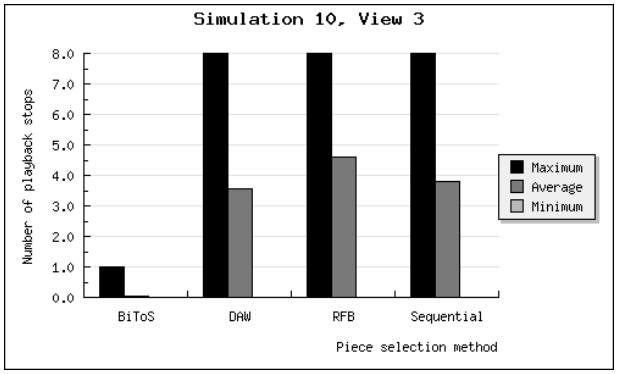

Figure 26. Number of playback stops for each peer when playback skips and stops, with peers joining at regular intervals.

should theoretically be possible for all peers without interruption. Instead, we get figures up to above 40% of all pieces skipped (for instance in Fig. 10) with average levels of more than 1/8 of all pieces skipped (for instance in Fig. 10 and 16). The situation for stops is not very good either, with an average of at least one stop for every 100 pieces played back (Fig. 14). Neither of these results is very good from an end-user point of view. The conclusion we can draw from this is that we need a better way of determining when to start, or restart, playback, as the methods we have used here do not seem to work in that regard. As for comparing the performance of DAW to the others, we note that in there three simulations DAW always comes up as the method that gives the best results with regard to playback position, without ever being the worst in any other regard.

One major concern regarding piece selection methods is whether they work well with the peer selection methods, in other words, whether the method of selecting pieces to request is compatible in practise with the methods used to determine which peers to send data to. The original tit-for-tat method from BitTorrent requires that data is exchanged both ways between peers in order to function well, and therefore is not a good solution when combined with the sequential method, where data is received from peers with more pieces and sent to peers with fewer pieces, exclusively. RFB would obviously work rather well in this fashion also, and the division of pieces into a high-priority set and a remaining pieces set as in BiToS is because the acquisition of pieces from the latter is "beneficial due to the Tit-for-Tat policy" [6]. We have not done any testing of the distance-availability weighted method on this subject, but our estimate is that it would be compatible. When the ratio of seeds to regular peers is low, the availability of pieces is important and the distance-availability weighted method selects pieces in a manner reminiscent of the rarest-first method, where the tit-for-tat policy has been proved as working. Conversely, when the ratio of seeds to regular peers is high, the distance-availability weighted method behaves similarly to the sequential method, but as the seeds do not require data to be sent to them, the need for out-of-order transfers in order to get the tit-for-tat policy working diminishes. We therefore believe our method would work in such a case as well.

VI. Related Work

This paper is an extended version of [1], which is a further development of a concept introduced in [14]. Whether due to its popularity or some other factor, such as its lack proprietarity, BitTorrent has been the subject of several other research projects during the last few years; some of which are directly relevant to ours.

The BitTorrent protocol has been analysed in real-world usage and found to be an efficient and viable solution for file sharing [8][15]. When it comes to the idea of on-demand streaming with BitTorrent as a basis, one of the more interesting propositions is the previously mentioned BiToS [6]. The main difference between BiToS and our distance-availability weighted method is that BiToS seems designed to maximise the amount of received pieces in a situation where the media bit rate is the same as the download rate of the client, while our solution is designed for a situation where the download bit rate is higher than the media bit rate, and all pieces should be received is such a way that playback is possible before all of the file has been transferred.

The approach used in BiToS is further expanded in [16], which also adds modifications to how the peers choose which peers to send data to, effectively replacing the tit-for-tat peer selection method used in BitTorrent with a method called Give-to-Get. The application Tribler uses Give-to-Get [17], while its protocol also contains other additions such as social networking. Another project combining social networking with BitTorrent and media streaming is

OneSwarm [10], adding "friend-to-friend" file sharing and, as of version 0.6 and later, the ability to choose between "streaming" media files (downloading sequentially) or not (downloading using rarest-first) [18]. As mentioned in Section I, there is also a project underway to "turn BitTorrent into a point-click-watch experience much more similar to YouTube", based on the popular, closed source BitTorrent application µTorrent [7].

Another BitTorrent-based service is LiveBT [19], which replaces the rarest-first method with Most-wanted-Block-Download-First (MBDF). In MBDF, a peer has a set of most wanted blocks (pieces), which is a fixed number of undownloaded pieces. Each peer also knows the most wanted blocks of its peers. With a probability *p* the peer's own most wanted piece is selected, and with probability 1-*p* the most wanted piece of its peers.

The idea of using a weighted priority is not unique to the DAW method. Wu et al [20] propose a weighted piece selection method, but for downloading instead of streaming. Their idea is that peers which hold many pieces are given greater weight than peers with few pieces when computing priorities. Whether this could improve performance also in streaming remains to be seen.

Besides BitTorrent-based solutions, there are also other projects underway to use peer-to-peer networks for on-demand streaming. Peer-to-peer networks serve as the backbone of both Spotify [21] and Voddler [22], the former a platform for on-demand streaming music, while the latter enables on-demand movies. Both these services distribute commercial content and therefore have limitations on usage as well as do not provide technical details on how their networks function.

VII. Conclusion and Future Work

In our first set of simulations, the distance-availability weighted piece selection method seems to be a better solution for on-demand streaming than either a straightforward sequential method or a modified version of the rarest-first method. Our second set of simulations, where in addition to the previously mentioned piece selection methods we also include the one presented in [6], do not show results contradicting the first set of simulations. However, the differences seem to be smaller than we previously thought, and none of the piece selection methods simulated seemed to be sufficiently good to work perfectly in the conditions given. However, as we have still only simulated the theoretical performance of the piece selection methods, we cannot comment on how they would work in a real-life environment. Future work on the subject could include practical implementation and real-world testing of the distance-availability weighted piece selection method, as well as comparison to other piece selection methods for streaming than the ones used for comparison here.

Acknowledgement

We thank Professor Kaisa Sere who has been very helpful during this project.

References

[1] P. Sandvik and M. Neovius, "The Distance-Availability Weighted Piece Selection Method for BitTorrent: A BitTorrent Piece Selection Method for On-Demand Streaming", In Proceedings of AP2PS '09, Sliema, Malta, October 2009

[2] B. Cohen, "BitTorrent - a new P2P app", Yahoo eGroups, http://finance.groups.yahoo.com/group/decentralization/message/3160 (Accessed August 2010)

[3] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy and M. Faloutsos, "File-sharing in the Internet: A characterization of P2P traffic in the backbone", Technical report, November 2003.

[4] Ipoque Internet Study 2008 / 2009. Available from http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009 (Accessed August 2010)

[5] B. Cohen, "Incentives Build Robustness in BitTorrent", In Proc. of IPTPS, 2003.

[6] A. Vlavianos, M. Iliofotou and M. Faloutsos, "BiToS: Enhancing BitTorrent for Supporting Streaming Applications", 9th IEEE Global Internet Symposium 2006 (in Conjunction with IEEE INFOCOM 2006).

[7] µTorrent Labs: Project Falcon, http://www.utorrent.com/labs/falcon (Accessed August 2010)

[8] A. Legout, G. Urvoy-Keller and P. Michiard, "Rarest First and Choke Algorithms Are Enough", In Proceedings of ACM SIGCOMM/USENIX IMC'2006, Rio de Janeiro, Brazil, October 2006.

[9] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy and A. Venkataramani, "Do incentives build robustness in BitTorrent?", 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 2007).

[10] T. Isdal, M. Piatek, A. Krishnamurthy and T. Anderson, "Friend-to-friend data sharing with OneSwarm", Technical report, UW-CSE, February 2009.

[11] D. Choffnes and D. Bustamante, "Taming the Torrent: A practical approach to reducing cross-ISP traffic in P2P systems", In Proceedings of ACM SIGCOMM 2008, August 2008.

[12] A. Nordberg, "Introduction to BitTorrent", Umeå University, 2006. Available from http://www.rasterbar.com/products/libtorrent/bittorrent.pdf (Accessed August 2010)

[13] B. Cohen, "The BitTorrent Protocol Specification", January 2008, http://www.bittorrent.org/beps/bep_0003.html (Accessed August 2010)

[14] P. Sandvik, "Adapting Peer-to-Peer File Sharing for On-Demand Media Streaming", Master of Science Thesis, Åbo Akademi University, May 2008.

[15] A. Legout, G. Urvoy-Keller and P. Michiardi, "Understanding BitTorrent: An Experimental Perspective", Technical Report (inria-00000156, version 2 - 19 July 2005), INRIA, Sophia Antipolis, July 2005.

[16] J.J.D. Mol, J.A. Pouwelse, M. Meulpolder, D.H.J. Epema and H.J. Sips, "Give-to-Get: Free-riding-resilient Video-on-Demand in P2P Systems", Proc. of SPIE, Multimedia Computing and Networking Conference (MMCN), vol. 6818, article 681804, 2008.

[17] A. Bakker et al, "Tribler Protocol Specification v0.0.2", January 2009, Available from http://www.tribler.org (Accessed August 2010)

[18] OneSwarm Changelog, http://wiki.oneswarm.org/index.php/Changelog (Accessed August 2010)

[19] J. Lv, X. Cheng, Q. Jiang, J. Ye, T. Zhang, S. Lin and L. Wang, "LiveBT: Providing Video-on-demand Streaming Service over BitTorrent Systems", pdcat, pp.501-508, Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007), 2007

[20] C. Wu, C. Li and J. Ho, "Improving the Download Time of BitTorrent-like Systems", IEEE International Conference on Communications 2007 (ICC 2007), Glasgow, Scotland, June 2007

[21] Spotify, http://www.spotify.com/fi/help/faq/tech/ (Accessed August 2010)

[22] Voddler, http://www.voddler.com/help/topic/2721821230584819787 (Accessed August 2010)