An Optimal Basis For Efficient Peer-To-Peer Content Distribution Algorithms

Jaakko Kangasharju
Helsinki Institute for Information Technology
Helsinki, Finland

Jussi Kangasharju
Technical University of Darmstadt
Darmstadt, Germany

Abstract— **Peer-to-peer content distribution has become extremely popular, thanks to its highly scalable performance. In this paper, we derive a lower bound on the performance of chunk-based peer-to-peer content distribution systems and develop an algorithm that is within 1 round of the lower bound in special cases, and within $1 + \log_2^2(I)$ rounds in the general case, where I is the number of peers. We consider the performance of our algorithm also in a heterogeneous bandwidth environment and under churn. We show that our algorithm always achieves good performance and does not impose an undue burden on fast peers, thus providing a natural incentive for all peers to participate.**

I. Introduction

Peer-to-peer content distribution is becoming the *de-facto* content distribution mechanism for distributing large files to a large number of interested users. Peer-to-peer content distribution has been shown to be an efficient and a highly scalable content distribution mechanism [3], [8]. In a peer-to-peer content distribution system large files are typically divided into small, constant-size chunks (e.g., 256 KB) and peers download chunks from other peers and allow others to download from them the chunks which the peer possesses [5].

In this paper we address the problem of selecting the chunk to download in a chunk-based peer-to-peer content distribution system. Specifically, we derive a lower bound on the distribution time of *any* chunk-based content distribution scheme and we present an algorithm which, in special cases, is within 1 distribution round of the lower bound, and, in the general case, is within $1 + \log_2^2(I)$ rounds of the lower bound.

A second, more important metric that we consider is the delay stretch which captures how much the download of an individual peer is slowed down by having to share chunks with other peers. As we argue in Section II-B, the delay stretch is a much more appropriate metric for heterogeneous situations than the commonly considered distribution time. To the best of our knowledge, no previous work has yet considered other metrics besides the distribution time.

We also investigate our algorithm in more realistic conditions, such as asymmetric and heterogeneous bandwidth conditions and peers joining and leaving the system during the distribution (also called churn). As our results show, our algorithm needs only minor changes to handle these cases and the effect on performance is negligible. Furthermore, our algorithm does not require any additional incentives, nor does it require that peers remain in the system after they have finished their downloads. Hence, the algorithm is very appropriate for heterogeneous environments with selfish peers, such as the Internet.

This paper is organized as follows. Section II presents the lower bound as well as algorithms for chunk-based content distribution. In Section III we consider the communication overhead of our algorithms. Section IV discusses how our algorithms behave under churn. In Section V we review related work. Finally, Section VI concludes the paper.

II. Optimal Algorithms

In this section we will derive bounds on the performance of any chunk-based content distribution system and develop an algorithm which we show to be very close to this bound.

A. Model

Assume that there are I peers interested in a file and the file has K chunks. Furthermore, assume that one peer (called the seed; also considered as peer number 1) has the complete file and other peers have no chunks in the beginning. We assume that there is an additional entity, called *tracker* (similar to the tracker in BitTorrent [5]) which helps us coordinate the downloading peers. The tracker maintains a list of all peers who are currently downloading the file. We also assume that time is discretized into *rounds*, and one round is defined as the time it takes for a peer to download or upload a chunk. In our model the chunks are *discrete*, i.e., a chunk is available for uploading only after it has been completely downloaded. A peer can upload and download in the same round. In Section II-F we discuss the implications of non-homogeneous bandwidth. In terms of peers joining and leaving, we first assume that all the peers join the file distribution at the same time and that no peers leave until the complete file has been distributed to all peers. In Section IV we consider how churn affects our algorithms. We also make the natural assumption that $K \geq \log_2(I)$, since typically the files being distributed are very large, hence the number of chunks will be very large.

B. Performance Metrics

There are two natural performance metrics that we consider. The first is the total time to distribute the complete file to all the clients. We call this metric *distribution time*. It is the time when the last client receives the last chunk of the file; other clients may have finished already much earlier. This metric has also been widely studied in literature.

The second metric we consider is what we call *delay stretch*. The delay stretch captures the increase in the time to download the complete file for an individual peer, compared to the case of the single peer just downloading the file from the seed (or fastest source). In other words, the delay stretch is a measure of how much overhead the distribution algorithm represents. For example, if a peer would get the file in K rounds by downloading directly from the seed and in $2K$ rounds using some distribution algorithm, the delay stretch would be 2.

We argue that the delay stretch is more appropriate, since distribution time as a metric has one significant problem in realistic scenarios. Minimizing the distribution time carries with it the implicit assumption that peers do not care when their *own* download finishes. This is because distribution time measures only when the last peer finishes, but does not give us any information about when the other peers have finished. While this assumption about altruistic peers might be true in some closed environments where cooperation can be enforced, it cannot be assumed to hold in Internet, where peers are known to behave selfishly [6]. In such cases, the delay stretch is more appropriate, since it describes how much the performance decreases because of cooperation. An algorithm that minimizes delay stretch is likely to be more acceptable, since the effect of "other peers" is minimized.

Note that in homogeneous network environments (all peers have the same bandwidth), the two metrics are identical, in the sense that an algorithm which achieves the lower bound on distribution time has also the lowest maximum delay stretch. Note that the maximum delay stretch may only apply to a few peers in any given case; other peers might still finish their downloads earlier. However, the maximum delay stretch gives us an overall idea of how well an algorithm performs.

In this paper we derive expressions for the maximum delay stretch of our algorithms. Investigating other possibilities, e.g., mean or median delay stretch, is part of our future work.

C. Lower Bound

We will now turn our attention to deriving the lower bound of the distribution time under homogeneous bandwidth conditions. As mentioned above, the lower bound also gives us the minimal maximum delay stretch.

Under assumptions from Section II-A, to distribute 1 chunk to all I peers takes $\lceil \log_2(I) \rceil$ rounds, since the number of peers holding a chunk can at most double in one round. Since the seed can send only 1 chunk per round, it will take at least $K-1$ rounds to put chunks $1, \ldots, K-1$ into distribution. After this, it will take at least $\lceil \log_2(I) \rceil$ rounds to distribute the last chunk to all the peers.

Hence, a lower bound R_{min} on the number of rounds for any chunk-based content distribution system is:

$$R_{min} = K - 1 + \lceil \log_2(I) \rceil. \tag{1}$$

Note that this lower bound applies to *any* chunk-based content distribution in the model of Section II-A, regardless of the details of the system.

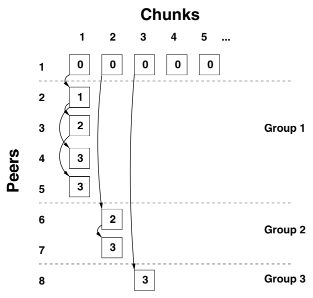

Fig. 1. Ramp-up phase. The squares are the chunks sent to peers and the number inside the square indicates in which step that chunk is transmitted. The arrows show which peer sends which chunk to which peer.

Consequently, the maximum delay stretch is

$$D_{min} = \frac{R_{min}}{K} = 1 + \frac{\lceil \log_2(I) \rceil - 1}{K} \tag{2}$$

since a peer could download the file in K rounds.

D. Number of Peers is 2^i

We initially make the assumption that I is a power of 2 to illustrate our algorithm better, but in Section II-E we relax this assumption and extend our algorithm to all values of I.

We divide the distribution of a file in three phases. In the *ramp-up phase*, fewer than I peers possess chunks, hence some peers cannot upload any chunks in the ramp-up phase. In the *distribution phase*, all I peers have at least one chunk and are uploading, but some chunks are only possessed by the seed. In the *endgame phase*, there is at least one copy of every chunk in addition to the copy held by the seed.

Ramp-up Phase

For the ramp-up phase, we divide the remaining $I-1$ peers $2, 3, \ldots, I$ into $\log_2(I)$ groups such that group i contains $I/2^i$ peers. This can be done by the tracker. Each peer is assumed to know in which group it belongs to and who are the other members of its group. The ramp-up phase is shown in Figure 1. In the ramp-up phase, we perform $\log_2(I)$ steps as follows. In step i, the seed sends chunk i to some member of group i. Groups $1, \ldots, i-1$ distribute the chunk they have received in an earlier step among themselves, as shown in Figure 1. For group i having $I/2^i$ members this distribution takes $\log_2(I)-i$ rounds. Since chunk i is placed into circulation on round i, all groups complete the ramp-up phase at the same time, i.e., on round $\log_2(I)$.

Distribution Phase

The second phase, the distribution phase, proceeds as follows. We classify chunks into three classes:

1) A chunk is *completed* if all the peers possess that chunk.
2) A chunk is *empty* if only the seed has that chunk.
3) Otherwise a chunk is called *active*.

The distribution phase lasts $K - \log_2(I)$ steps, and at any given time, there are $\log_2(I)$ active chunks. As in the ramp-up phase, these active chunks are distributed among the groups so that each group has a single active chunk that is not possessed by any peer in the other groups. We number the groups so that the active chunk of group i at the start of step k is $k+i-1$.

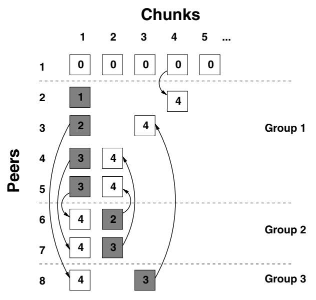

Fig. 2. Distribution phase. The gray chunks were distributed in the ramp-up phase and the white chunks are distributed in the distribution phase.

The actual steps for $k = 1, \ldots, K - \log_2(i)$ are:

1) Select any peer from group 1. This peer does not send a chunk, but receives chunk $k + \log_2(I)$ from the seed.
2) All other $I/2 - 1$ members of group 1 send chunk k (which they all possess) to all other peers in the other groups. There are $I/4 + \cdots + 1 = I/2 - 1$ such peers.
3) A member in group $i = 2, \ldots, \log_2(I)$ sends chunk $k + i - 1$ to that member of group 1 who sent a chunk to it.
4) Re-create the groups such that the distribution of chunks to the groups corresponds to the distribution at the end of the ramp-up phase, with the exception that chunk k has been completed and chunk $k + \log_2(I)$ has become active.

Figure 2 shows step 1 of the distribution phase, i.e., step 4 of the algorithm. We re-organize the groups such that peers 4, 5, 6, and 7 are in the same group, peers 3 and 8 in one group, and peer 2 is in the last group. We will discuss this organization in detail in Section III, where we will derive a bound on the number of peers that a single peer must communicate with.

After $K - \log_2(I)$ steps in the distribution phase, all chunks are either completed or active, which leads us to the last phase.

Endgame Phase

At the start of the endgame phase, there are $\log_2(I)$ active chunks. Sending chunks the same way as in the distribution phase (leaving unnecessary operations out), we can complete one active chunk per step. The length of the endgame phase is therefore $\log_2(I)$. Note that even though some peers are not uploading any chunks, all the peers still missing chunks are downloading on every round, hence the additional upload capacity cannot be used and the length remains $\log_2(I)$.

The total length of the distribution is $(\log_2(I)) + (K - \log_2(I)) + \log_2(I) = K + \log_2(I)$ steps. Comparing the this to the lower bound from equation (1), we can see that our algorithm needs only 1 round more to distribute the chunks.

The delay stretch of our algorithm is then $1 + \log_2(I)/K$, i.e., only $1/K$ larger than the lower bound. As the number of chunks increases our algorithm gets closer and closer to the lower bound, thus improving performance.

E. Arbitrary Number of Peers

In this section we will extend the algorithm to support the case where the number of peers is arbitrary. In order to achieve this, we need to change the way the groups are formed. We will divide the peers into groups, such that each group has $2^n - 1$ peers for some value of n. We call such groups "cliques", and continue to use the term "group" for a part of a clique having 2^i members for some i. We denote the number of cliques by C and the total number of groups in all cliques by G. A clique with $2^n - 1$ peers is called an n-clique.

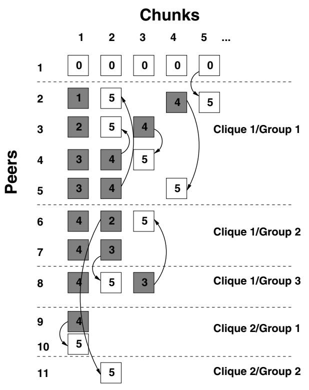

Fig. 3. Round 5 of ramp-up phase for 2 cliques

The standard greedy coin changing algorithm minimizes both C and G [4]. From this it follows that if there are two n-cliques in this optimal division, there cannot be any additional k-cliques with $k \leq n$. Therefore C is bounded above by $\lfloor \log_2(I) \rfloor + 1$ and G by $1 + \sum_{k=1}^{\lfloor \log_2(I) \rfloor} k = 1 + (\lfloor \log_2(I) \rfloor^2 + \lfloor \log_2(I) \rfloor)/2 \leq 1 + \log_2^2(I)$. For example, if there are 22 peers, then we have cliques of 15, 3, and 3 peers, with the remaining 1 peer being the seed. In this case, $C = 3$ and $G = 8$.

Figure 3 shows a network of 11 peers, divided into 2 cliques of 7 and 3 peers. The ramp-up phase up to round 3 proceeds exactly the same way as in the algorithm with 2^i peers, shown in Figure 1. In the 4th round, shown in Figure 2, peer 2 does not upload any chunks. We can use this additional upload capacity to send chunk number 1 to a peer in the biggest group of clique 2, i.e., peer 2 sends chunk 1 to peer 9 (or peer 10). For clique 2, peer 2 appears as the seed.

In round 5, clique 1 continues distribution as usual in a group of 8 peers, with one of the peers 4, 5, 6, or 7 not having to upload any chunk within the clique (peer 6 in the case of Figure 3). That peer would send chunk 2 to peer number 11, thus seeding chunk 2 in that clique. On round 5 in clique 2, peer 9 would send chunk 1 to peer 10. Since every peer now has at least one chunk, the ramp-up phase would be completed.

As we see from the above example, the ramp-up phase in the general case is longer than in the case with 2^i peers. This is due to the chunks having to "trickle" through all the cliques to the last group of the last clique. Whereas in the 2^i case, the ramp-up phase takes $\log_2(I)$ rounds, the number of rounds required in the general case is G, i.e., 5 rounds in our example.

However, the distribution phase is correspondingly shorter, and the total length of the ramp-up and distribution phases remains K. This can easily be verified, since on every round during ramp-up and distribution, one chunk is transformed from empty to active, thus needing K rounds for all K chunks to become active.

As there are G active chunks at the end of the distribution phase and the endgame phase completes one chunk on each round, the endgame phase will last $G \leq 1 + \log_2^2(I)$ rounds.

Hence, the generalized version of our chunk selection algorithm can distribute all the chunks to all the peers in at most $K + 1 + \log_2^2(I)$ rounds. This implies a delay stretch of $1 + (1 + \log_2^2(I))/K$. As with the basic algorithm, when the number of chunks increases, the delay stretch also diminishes proportionally to the number of chunks.

F. Heterogeneous Bandwidth

The algorithm above assumes that the bandwidth of the peers is homogeneous and that we can upload and download only 1 chunk per round. We now discuss how the algorithms are affected by realistic bandwidth conditions.

If the bandwidths of the peers are homogeneous, but asymmetric, we have two possibilities. If the upload bandwidth is smaller than the download bandwidth, then all peers are already uploading chunks whenever possible in the ramp-up and distribution phases. Hence, the upload capacity is already at its maximum and the increased download bandwidth remains unused. The only exception is in the endgame phase, where peers are able to download chunks from several sources, hence the endgame phase could be slightly shorter. If the download bandwidth is smaller, then in the distribution and endgame phases, all peers (that are still downloading) are downloading on every round, hence the additional upload capacity cannot be used. Additional upload capacity could make the ramp-up phase shorter.

From the above, we can conclude that any homogeneous bandwidth case even with asymmetric upload and download bandwidths at most shortens the ramp-up or endgame phases proportionally to the bandwidth asymmetry. Likewise, the delay stretch is only marginally affected in these cases.

If the peer bandwidths are heterogeneous, that is, each peer has different upload and download bandwidths, deriving a lower bound becomes more complicated. If we normalize all the bandwidths by the download bandwidth of the slowest peer, the distribution will take at least K such normalized rounds, since that is the time it takes for the slowest peer to obtain all the chunks. Other peers could complete the download much faster, but the complete distribution would be bounded by K. However, this does not tell us anything about the delay stretch for the faster peers.

To minimize the delay stretch for the faster peers, the correct solution is to divide the peers into cliques *according to their bandwidths*, by attempting to have all the peers in a clique have roughly the same bandwidth. This may result in the number of cliques being larger than the minimum required, which will slightly increase the distribution time for the last cliques. These would typically be the slowest peers, hence the additional time would not matter much. However, by having all the peers in a clique have roughly the same bandwidth, they are able to perform the distribution within the clique much faster than if there were also slow peers in the clique. Hence the delay stretch will be smaller, on the order of $1 + n/K$ for the fastest n-clique. In other words, the faster peers are not affected by the slower peers and experience performance which they would get in a system with only fast peers. Note that since a round is defined as the time to download one chunk, different cliques would have rounds that are of different lengths.

Since the chunks trickle through the cliques, the distribution time of a clique is actually dominated by the download bandwidth of the *next* clique to which the chunks trickle from the current clique. Assuming that there are no large "jumps" in the bandwidths, this is likely to have only a negligible effect on the overall performance of the system.

Note that a clique with high-bandwidth peers will finish their download quickly. They can then serve as additional seeds for the slower peers, thus improving the distribution time and delay stretch for the slower peers. However, the algorithm does not rely on peers remaining in the system after completion; the slow peers will finish in the number of rounds indicated in the worst case and if some peers altruistically remain in the system, the slow peers can finish faster.

III. Communication Overhead

In this section, we will turn our attention to the number of peers that a given peer must communicate with during the distribution phase. First, we present an algorithm that determines the composition of the different groups during the distribution phase. We present only the case where the number of peers I to be a power of 2; extension to multiple cliques is straightforward. The message overhead of the algorithm is minimal, since the initial coordination is done by the tracker and the peers can run the algorithm independently.

A. Group Assignment

Under the above assumptions, we have I peers and $\log_2(I)$ groups of peers, such that group i has $I/2^i$ peers. We label each peer with its initial group number. We start counting the rounds from 0.

Intuitively, the algorithm can be described as follows. In any given round, all peers which are in group $i > 1$ will be in group $i - 1$ in the following round. All the empty places in the other groups will be filled with peers that were in group 1 in that round. Half of the peers who were in group 1 will remain in group 1. The key to the algorithm is to select which peers from group 1 leave that group and into which groups they are assigned. We use the following procedure for determining this.

1) On any given round, half of the peers in group 1 are labeled with 1, and the rest have labels 2,…,$\log_2(I)$. We call the peers with a given label a *set*. From each of the sets, we select half of the peers arbitrarily. Set $i = 2$.
2) If the smallest set has only 1 peer, go to Step 4, otherwise select half of the peers from each set and put them in group i.
3) Increase i by 1 and repeat step 2 with the remaining peers in the sets.
4) When the smallest set in Step 2 has only 1 peer, distribute the remaining peers in the order of their labels into the remaining free places in the groups.

Round 0:

1_1 1_2 1_3 1_4 1_5 1_6 1_7 1_8
2_9 2_{10} 2_{11} 2_{12}
3_{13} 3_{14}
4_{15}

Round 1:

1_1 1_2 1_3 1_4 2_9 2_{10} 2_{11} 2_{12}
1_5 1_6 3_{13} 3_{14}
1_7 4_{15}
1_8

Round 2:

1_1 1_2 1_5 1_6 2_9 2_{10} 3_{13} 3_{14}
1_3 1_7 2_{11} 4_{15}
1_4 1_8
2_{12}

Round 3:

1_1 1_3 1_5 1_7 2_9 2_{11} 3_{13} 4_{15}
1_2 1_4 1_6 1_8
2_{10} 2_{12}
3_{14}

Fig. 4. Group assignment algorithm

On the last round (round $\log_2(I) - 1$), we simply select all the peers with a label other than 1, in order to return to the original situation after $\log_2(I)$ rounds.

Figure 4 illustrates the algorithm for $I = 16$. The seed is not shown, so only 15 peers are shown. The labels are of the form i_p, where i is the label (original group number) and p is the number of the peer, where we assume that all peers have initially been numbered, starting from 1.[1] Note that after 4 rounds, the situation would revert to the initial situation.

B. Inter-Peer Communication

Given the above algorithm for group assignment, we now calculate the number of peers that a given peer must communicate with during the distribution phase.

We assume that each peer knows its number and group number at the beginning, for example by ordering the peers according to their IP addresses. Recall that the tracker knows every peer and can easily provide this information. Therefore there is no need for additional coordination between the peers.

As we have shown, the assignment of peers to groups follows a cycle of $\log_2(I)$ rounds. This implies that a peer needs to communicate only with the peers that it communicates with during one cycle. Recall from Section II-D that during the distribution phase, peers form pairs to exchange chunks. This means that on every round, a peer needs to communicate with exactly one peer. Since there are $\log_2(I)$ rounds in the cycle, this means that a peer will exchange chunks with $\log_2(I)$ peers during a cycle. Because the cycle repeats, the peer can exchange chunks with the exactly same peers during the next cycle. Hence, a peer needs to know only $\log_2(I)$ other peers.

To consider the more general case where the number of peers is arbitrary, Section III-A shows that it is possible to arrange intra-clique distribution in an n-clique so that the group assignment is restored after n rounds. Therefore, by repeating the same distribution pattern throughout the algorithm it is necessary for each peer to communicate only with n peers in its own clique. Finally, on each round one of the peers in group 1 of a clique sends to a different peer from the one it receives from, so it communicates with one additional peer, and so the required connectedness of the distribution graph is bounded above by $n + 1 \leq \lfloor \log_2(I) \rfloor + 1$.

[1]This number is shown for illustration; the algorithm does not need it.

IV. Handling Churn

In the algorithms presented so far, we have assumed that all the peers arrive at the same time at the beginning of the file distribution, no new peers join during the distribution, nor do any peers leave prematurely. We will now discuss how churn affects our algorithms.

A. Peers Joining

New peers joining the system during the distribution are only a small problem. We take all the peers that arrive during a round and the tracker forms a clique out of them and adds it as “the last clique”. If peers join the system frequently, it might be beneficial to wait a few rounds before forming the new clique, so that we are able to create a larger clique. This is because each clique is a slow-down of 1 round to all subsequent cliques, hence minimizing the number of cliques is beneficial to all.

Note that if peers have heterogeneous bandwidths, we would form possibly several new cliques, in order to group peers with similar bandwidths in the same clique. This would be a further incentive to wait a few rounds before forming the new cliques. Since the files are typically large, the overhead of having to wait a few rounds is offset by the better distribution time and delay stretch. The tradeoff between how many peers are needed for a new clique and how long to wait to get a bigger clique, is part of our future work.

B. Peers Departing

The case of peers departing is more complicated. First, assume that only one peer per clique leaves during a given round. In this case, the clique has $2^n - 2$ peers remaining (for some n) and can be divided into two $n-1$-cliques, which then continue the distribution as usual. Peers in one of the new cliques will not see any effect on their performance, whereas peers in the other new clique will experience a slowdown of 1 round. The departure will slow down the distribution time for all peers in that and subsequent cliques by one round. The worst case is that the departing peer is the peer that just received a new chunk. Then, one additional round is needed to get that chunk again.

It may seem that frequent peer departures might have a significant, adverse effect on performance, since each departure will add one round to each of the “downstream” cliques. However, in practice, we can merge many of the new cliques formed by departures to form larger cliques (see below for details on merging of cliques). In this case, we try to arrange the new cliques according to bandwidth and progress in the distribution, so that the new, larger cliques can continue the distribution in the most efficient manner possible. Some peers will experience an increased delay stretch, since they might be put in the same clique with slower peers, but other peers will experience considerably shorter distribution times (and delay stretches) compared to the situation where we would leave the small cliques intact.

C. Merging Cliques

Formally, the merging of cliques happens as follows. Assume two n-cliques A and B and that A is "ahead" of B, i.e., A has more chunks than B. This can easily be verified by the tracker. If a new peer N joins, we can merge cliques A and B with N to form one $n+1$-clique. In the new clique, the peers from the old clique B and N will form group 1. Peers from the old clique A which were in group i in A, will now be in group $i+1$.

Then, the distribution proceeds as per the algorithm, with the exception that peers who were in B (and peer N) do not upload anything to A, since the peers in A already have all the chunks that peers in B have. As a result, the peers who were in B will have a gap in their chunks, since they will start to get new chunks from A and have to start distributing those. This gap will then be filled during the endgame phase which will require that either one peer from A remains to send the missing chunks to B or that they get them from the seed. Naturally, the new peer N will have a gap from the beginning and this needs to be filled also at the end.

Investigating this more closely and identifying under which conditions should new cliques be formed, as well as the overhead incurred for merging, is part of our future research.

V. Related Work

One of the most popular chunk-based content distribution systems is BitTorrent. For a detailed description of how BitTorrent works, please see [5]. Although we do not explicitly model or analyze BitTorrent in this paper, we base some of our assumptions (one seed, tracker-like functionality for knowing which peers are downloading) on it. The good performance of BitTorrent shows that such assumptions are realistic for building a practical, large-scale system.

Work in [1] considers a problem similar to ours. The main differences between our work and theirs is that our lower bound on distribution time is much tighter. They only consider a trivial lower bound of K rounds. Furthermore, their results, including their lower bound, appear to be the results of simulations, whereas our work can be provably derived from a small set of core assumptions. The results from [1] actually confirm our theoretical results from this paper.

Biersack et al. [2] present several different algorithm for a chunk-based content distribution system. Their approaches are similar to ours. Their work does not address the issue of a lower bound, nor are their algorithms as general as the ones in this paper, since they require a specific number of children per peer. Their best performing algorithm $PTree^k$ requires that each peer has k children; our generalized algorithm from Section II-E imposes no limitations on the number of peers and achieves the same, or better performance as $PTree^k$.

Work in [3], [8] considers the capacity of peer-to-peer content distribution networks to serve traffic. In [3] different chunk selection algorithms are compared in homogeneous environments. They also claim that the differences between algorithms are smaller in heterogeneous environments. Yang and de Veciana [8] discuss an algorithm similar to Section II-D, but their focus is on deriving expressions for the service capacity of peer-to-peer networks and they do not consider general case of arbitrary number of peers.

Schiely et al. [7] perform simulations and experiments in a heterogeneous bandwidth environment using tree-based distribution. Their results show that it is advantageous to organize the high-bandwidth peers higher in the tree, similar to organizing the peers into cliques according to their bandwidths (as in Section II-F). Their work only covers a static case and does not consider what happens under churn.

What is common to all the previous work mentioned above, is that they only consider distribution time as a metric. As argued in Section II-B, distribution time is only appropriate in homogeneous situations, and in heterogeneous situations a metric like the delay stretch should be used.

VI. Conclusion

In this paper, we have considered optimal distribution algorithms for peer-to-peer content distribution. We have derived a lower bound which applies to any chunk-based content distribution system. We have also developed an algorithm which gets to within 1 round of the optimal in special cases and within $1+\log_2^2(I)$ in the general case. The delay stretch of our algorithm is also close to the minimal stretch. Our algorithm does not require excessive communication; each peer needs to communicate with $1+\log_2(I)$ other peers. We have also considered heterogeneous bandwidth environments as well as churn, and have shown that our algorithm needs only slight modifications to handle them and that the decrease in performance is negligible.

In our future work, we will implement the algorithms from this paper and perform a numerical evaluation of them. Our emphasis will be on evaluating the algorithms in heterogeneous bandwidth conditions.

References

[1] A. Al Hamra and P. A. Felber. Design choices for content distribution in P2P networks. *Computer Communications Review*, 35(5):29–40, Oct. 2005.

[2] E. W. Biersack, P. Rodriguez, and P. Felber. Performance analysis of peer-to-peer networks for file distribution. In *Fifth International Workshop on Quality of Future Internet Services*, Barcelona, Spain, Sept. 2004.

[3] P. A. Felber and E. W. Biersack. Self-scaling networks for content distribution. In *Proceedings of Self-**, Bertinoro, Italy, May 2004.

[4] T. C. Hu and M. L. Lenard. Optimality of a heuristic solution for a class of knapsack problems. *Operations Research*, 24(1):193–196, Jan.–Feb. 1976.

[5] M. Izal et al. Dissecting BitTorrent: Five months in a torrent's lifetime. In *Proceedings of PAM*, Apr. 2004.

[6] S. Saroiu, K. P. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing networks. In *MMCN*, San Jose, CA, Jan. 2002.

[7] M. Schiely, L. Renfer, and P. Felber. Self-organization in cooperative content distribution networks. In *Network Computing and Applications*, Cambridge, MA, July 2005.

[8] X. Yang and G. de Veciana. Service capacity of peer-to-peer networks. In *Infocom*, Hong Kong, Mar. 2004.