Optimal Scheduling of Peer-to-Peer File Dissemination

Jochen Mundinger[*], Richard Weber[†] and Gideon Weiss[‡][§]

Abstract

Peer-to-peer (P2P) overlay networks such as BitTorrent and Avalanche are increasingly used for disseminating potentially large files from a server to many end users via the Internet. The key idea is to divide the file into many equally-sized parts and then let users download each part (or, for network coding based systems such as Avalanche, linear combinations of the parts) either from the server or from another user who has already downloaded it. However, their performance evaluation has typically been limited to comparing one system relative to another and typically been realized by means of simulation and measurements. By contrast, we provide an analytic performance analysis that is based on a new uplink-sharing version of the well-known broadcasting problem. Assuming equal upload capacities, we show that the minimal time to disseminate the file is the same as for the simultaneous send/receive version of the broadcasting problem. For general upload capacities, we provide a mixed integer linear program (MILP) solution and a complementary fluid limit solution. We thus provide a lower bound which can be used as a performance benchmark for any P2P file dissemination system. We also investigate the performance of a decentralized strategy, providing evidence that the performance of necessarily decentralized P2P file dissemination systems should be close to this bound and therefore that it is useful in practice.

1 Introduction

Suppose that M messages of equal length are initially known only at a single source node in a network. The so-called *broadcasting problem* is about disseminating these M messages to a population of N other nodes in the least possible time, subject to capacity constraints along the links of the network. The assumption is that once a node has received one of the messages it can participate subsequently in sending that message to its neighbouring nodes.

1.1 P2P file dissemination background and related work

In recent years, overlay networks have proven to be a popular way of disseminating potentially large files (such as a new software product or a video) from a single server S to a potentially large group of N end users via the Internet. A number of algorithms and protocols have been suggested, implemented and studied. In particular, much attention has been given to

[*]EPFL-IC-LCA, BC203, Station 14, CH-1015 Lausanne, Switzerland. Email: jochen.mundinger@epfl.ch.

[†]Statistical Laboratory, Centre for Mathematical Sciences, Wilberforce Road, Cambridge CB3 0WB, UK. Email: rrw1@cam.ac.uk

[‡]Department of Statistics, University of Haifa, Mount Carmel 31905, Israel. Email: gweiss@stat.haifa.ac.il.
Research supported in part by Israel Science Foundation Grants 249/02 and 454/05.

[§]Collaboration of the authors was supported in part by European Network of Excellence Euro-NGI.

peer-to-peer (P2P) systems such as BitTorrent [7], Slurpie [29], SplitStream [5], Bullet' [18] and Avalanche [10], to name but a few. The key idea is that the file is divided into M parts of equal size and that a given user may download any one of these (or, for network coding based systems such as Avalanche, linear combinations of the bits in the file parts) either from the server or from a peer who has previously downloaded it. That is, the end users collaborate by forming a P2P network of peers, so they can download from one another as well as from the server. Our motivation for revisiting the broadcasting problem is the performance analysis of such systems.

With the BitTorrent protocol[1], for example, when the load on the server is heavy, the protocol delegates most of the uploading burden to the users who have already downloaded parts of the file, and who can start uploading those parts to their peers. File parts are typically 1/4 megabyte (MB) in size. An application helps downloading peers to find each other by supplying lists of contact information about randomly selected peers who are also downloading the file. Peers use this information to connect to a number of neighbours. A full description can be found in [7]. The BitTorrent protocol has been implemented successfully and is deployed widely. A detailed measurement study of the BitTorrent system is reported in [26]. According to [25], BitTorrent's share of the total P2P traffic has reached 53% in June 2004. For recent measurements of the total P2P traffic on Internet backbones see [17].

Slurpie [29] is a very similar protocol, although, unlike BitTorrent, it does not fix the number of neighbours and it adapts to varying bandwidth conditions. Other P2P overlay networks have also been proposed. For example see SplitStream [5] and Bullet' [18].

More recently, Avalanche[2] [10], a scheme based on network coding [1] has been suggested. Here, users download linear combinations of the bits in the file parts rather than individual file parts. This ensures that users do not need to find specific parts in the system, but that any upload by a given user can be of interest to any peer. Thus, network coding can improve performance in a decentralized scenario. Our results apply to any P2P file dissemination system, whether or not it uses network coding.

Performance analysis of P2P systems for file dissemination has typically been limited to comparing one system relative to another and has typically been realized by means of simulation and measurements. As an alternative method of performance analysis, we find the minimum makespan, that is the minimal time required to fully disseminate a file of M parts from a server to N end users in a centralized scenario. We thereby provide a lower bound which can be used as a performance benchmark for any P2P file dissemination system. We also investigate what loss in efficiency is due to the lack of centralized control. Using both theoretical analysis, simulation, and direct computation, we show that even a naive randomized strategy can disseminate the file in an expected time whose size grows with N in at a similar manner as the minimal expected time that could be achieved a centralized controller. This suggests that the performance of necessarily decentralized P2P file dissemination systems can be close to that of our performance bound and so this bound provides a practically useful as a benchmark.

In this paper, we provide the scheduling background, proofs and discussion of the results in our extended abstracts [24] and [23]. It is essentially Chapter 2 of [21], but we have added Theorem 5 and the part on theoretical bounds in Section 6. In [31] the authors also consider

[1]http://bitconjurer.org/BitTorrent/protocol.html
[2]http://www.research.microsoft.com/∼pablo/avalanche.aspx

problems concerned with the service capacity of P2P networks, however, they only give a heuristic argument for the makespan with equal upload capacities when N is of the simple form $2^n - 1$. In [27] a fluid model for BitTorren*t*-like networks is introduced and studied, also looking at the effect of incentive mechanisms to address free-riding. Link utilization and fairness are issues in [3]. In [20], also motivated by the BitTorrent protocol and file swarming systems in general, the authors consider a probabilistic model of coupon replication systems. Multi-torrent systems are discussed in [11]. There is other related work in [28].

1.2 Scheduling background and related work

The broadcasting problem has been studied in the context of many different network topologies. Comprehensive surveys can be found in [13] and [14]. The problem was first solved for a complete graph in [6] and [8], for a unidirectional telephone model in which each node can either send or receive one message during each round, but not do both. The minimal number of rounds required is $2M - 1 + \lfloor \log_2 (N+1) \rfloor$ for even N, and $2M + \lfloor \log_2 (N+1) \rfloor - \lfloor \frac{M-1+2^{\lfloor \log_2 (N+1) \rfloor}}{(N+1)/2} \rfloor$ for odd N.[3]

In the bidirectional telephone model nodes can both send and receive one message simultaneously, but nodes must be matched pairwise. That is, in each round, a node can only receive from the same node to which it is sending. The authors of [2] provide an algorithm which when N is odd takes $M + \lfloor \log_2 N \rfloor$ rounds and is optimal, and which when N is even takes $M + \lfloor \log_2 N \rfloor + M/N + 2$ rounds and is optimal up to an additive term of 3.

The simultaneous send/receive model of [19] relaxes the pairing constraint of the bidirectional telephone model and supposes that a node can send a message to a different node than the one from which it is presently receiving a message. The optimal number of rounds turns out to be $M + \lfloor \log_2 N \rfloor$. We will return to this result in Section 3.

In all the work reviewed above work the aim was to model interactions of processors, and so the authors found it natural to assume that all nodes have equal upload capacities (i.e., equal constraints on the rates at which they can send data). In this paper, we are interested in P2P file dissemination and so we permit nodes to have different upload capacities. We work with a new uplink-sharing model that is designed to model this (cf. Section 2). It is closely related to the simultaneous send/receive model, but is set in continuous time. Our work also differs in that we are motivated by the evaluation of necessarily decentralized P2P file dissemination algorithms, i.e., ones that can be implemented by the users themselves, rather than by some centralized controller. We retain an interest in the centralized case as a basis for comparison and to provide a bound on what can be achieved. We show that when upload capacities are equal the minimal number of rounds required is $M + \lfloor \log_2 N \rfloor$, just as for the simultaneous send/receive model. For general upload capacities, we provide two complementary solutions and investigate the performance of a decentralized strategy.

1.3 Outlook

The rest of this paper is organized as follows. In Section 2 we introduce the uplink-sharing model and relate it to the simultaneous send/receive model. Our optimal algorithm for the simultaneous send/receive broadcasting problem is presented in Section 3. We show that it

[3]Bar-Noy, Kipnis and Schieber report a slightly different expression in [2]. This appears to be a transcription error in quoting the result of Cockayne and Thomason.

also solves the problem for the uplink-sharing model with equal capacities. In Section 4 we show that the general uplink-sharing model can be solved via a finite number of mixed integer linear programming (MILP) problems. This approach is suitable for a small number of file parts M. We provide additional insight through the solution of some special cases. We then consider a limiting case in which the file can be divided into infinitely many parts and provide the centralized fluid solution. We extend these results to an even more general situation in which different users might have different (disjoint) files of different sizes to disseminate (Section 5). This approach is suitable for typical and for large numbers of file parts M. Finally, we address decentralized algorithms. In Section 6 we evaluate the performance of a very simple and natural randomized strategy, theoretically, by simulation, and by direct computation. Assuming that the peers have equal upload capacities, we consider two different information scenarios, and show that even this naive algorithm can disseminate the file in an expected time whose size grows with N at a similar rate as the minimal time that can be achieved by a centralized controller. This suggests that the performance of necessarily decentralized P2P file dissemination systems can come close to the performance bounds of the previous sections, and so these bounds provide practically useful benchmarks. We conclude and present ideas for further research in Section 7.

2 The Uplink-Sharing Model

We now introduce an abstract model for the file dissemination scenario described in the previous section, focusing on the important features of P2P file dissemination.

Underlying the file dissemination system is the Internet. Thus, each user can connect to every other user and the network topology is a complete graph. The server S has upload capacity C_S and the N peers have upload capacities $C_1, \dots, C_N$, measured in megabytes per second (MBps). Once a user has received a file part it can participate subsequently in uploading it to its peers (source availability). We suppose that, in principle, any number of users can simultaneously connect to the server or another peer, the available upload capacity being shared equally amongst the open connections (fair sharing). Taking the file size to be 1 MB, this means that if n users try simultaneously to download a part of the file (of size $1/M$) from the server then it takes n/MC_S seconds for these downloads to complete. Observe that the rate at which an upload takes place can both increase and decrease during the time of an upload (varying according to the number of other uploads with which it shares the upload capacity), but we assume that uploads are not interrupted until complete, that is the rate is always positive (continuity). In fact, Lemma 1 below shows that the makespan is not increased if we restrict the server and all peers to carry out only a single upload at a time. We permit a user to download more than one file part simultaneously, but these must be from different sources; only one file part may be transferred from one user to another at the same time. We ignore more complicated interactions and suppose that the upload capacities, $C_S, C_1, \dots, C_N$, are the only constraints on the rates at which file parts can be transferred between peers. This is a reasonable assumption if the underlying network is not overloaded. Finally, we assume that rates of uploads and downloads do not constrain one another.

The assumption that the download rates are unconstrained might be thought to be unrealistic. However, we show in Section 3 that if the upload capacities are equal then additional download capacity constraints do not increase the minimum possible makespan, as long as these download capacities are at least as big. This is usually the case in practice.

Typically, N is the order of several thousands and the file size is up to a few gigabytes (GB), so that there are several thousand file parts of size 1/4 MB each.

Finding the minimal makespan looks potentially very hard as upload times are interdependent and might start at arbitrary points in time. However, the following two observations help simplify it dramatically. As we see in the next section, they also relate the uplink-sharing model to the simultaneous send/receive broadcasting model.

Lemma 1 *In the uplink-sharing model the minimal makespan is not increased by restricting attention to schedules in which the server and each of the peers only carry out a single upload at a time.*

Proof. Identify the server as peer 0 and, for each $i = 0, 1, \dots, N$ consider the schedule of peer i. We shall use the term *job* to mean the uploading of a particular file part to a particular peer. Consider the set of jobs, say J, whose processing involves some sharing of the upload capacity C_i. Pick any job, say j, in J which is last in J to finish and call the time at which it finishes t_f. Fair sharing and continuity imply that job j is amongst the last to start amongst all the jobs finishing before or at time t_f. To see this, note that if some job k were to start later than j, then (by fair sharing and continuity) k must receive less processing than job j by time t_f and so cannot have finished by time t_f. Let t_s denote the starting time of job j.

We now modify the schedule between time t_s and t_f as follows. Let K be the set of jobs with which job j's processing has involved some sharing of the upload capacity. Let us re-schedule job j so that it is processed on its own between times $t_f - 1/C_iM$ and t_f. This consumes some amount of upload capacity that had been devoted to jobs in K between $t_f - 1/C_iM$ and t_f. However, it releases an exactly equal amount of upload capacity between times t_s and $t_f - 1/C_iM$ which had been used by job j. This can now be allocated (using fair sharing) to processing jobs in K.

The result is that j can be removed from the set J. All jobs finish no later than they did under the original schedule. Moreover, job j starts later than it did under the original schedule and the scheduling before time t_s and after time t_f is not affected. Thus, all jobs start no earlier than they did under the original schedule. This ensures that the source availability constraints are satisfied and that we can consider the upload schedules independently. We repeatedly apply this argument until set J is empty. ■

Using Lemma 1, a similar argument shows the following result.

Lemma 2 *In the uplink-sharing model the minimal makespan is not increased by restricting attention to schedules in which uploads start only at time* 0 *or at times that other uploads finish.*

Proof. By the previous Lemma it suffices to consider schedules in which the server and each of the peers only carry out a single upload at a time. Consider the joint schedule of all peers $i = 0, 1, \dots, N$ and let J be the set of jobs that start at a time other than 0 at which no other upload finishes. Pick a job, say j, that is amongst the first in J to start, say at time t_s. Consider the greatest time t_f such that $t_f < t_s$ and t_f is either 0 or the time that some other upload finishes and modify the schedule so that job j already starts at time t_f.

The source availability constraints are still satisfied and all uploads finish no later than they did under the original schedule. Job j can be removed from the set J and the number of jobs in J that start at time t_s is decreased by 1, although there might now be more (but at most N in total) jobs in J that start at the time that job j finished in the original schedule. But this time is later than t_s. Thus, we repeatedly apply this argument until the number of jobs in J that start at time t_s becomes 0 and then move along to jobs in J that are now amongst the first in j to start at time $t'_s > t_s$. Note that once a job has been removed from J, it will never be included again. Thus we continue until the set J is empty. ■

3 Centralized Solution for Equal Capacities

In this section, we give the optimal centralized solution of the uplink-sharing model of the previous section when upload capacities are equal. We first consider the simultaneous send/receive broadcasting model in which the server and all users have upload capacity of 1. The following theorem provides a formula for the minimal makespan and a centralized algorithm that achieves it is contained in the proof.

Our result agrees with that of Bar-Noy, Kipnis and Schieber [2], who obtained it as a by-product of their result on the bidirectional telephone model. However, they required pairwise matchings in order to apply the results from the telephone model. For the simultaneous send/receive model, they use perfect matching in each round for odd N, and perfect matching on $N-2$ nodes for even N. As a result, their algorithm differs for odd and even N and is substantially more complicated to describe, implement and prove to be correct than the one we present within the proof of Theorem 1. Theorem 1 has been obtained also by Kwon and Chwa [19], via an algorithm for broadcasting in hypercubes. By contrast, our explicitly constructive proof makes the structure of the algorithm very easy to see. Moreover, it makes the proof of Theorem 3, for the uplink-sharing model, a trivial consequence (using Lemmas 1 and 2).

Essentially, the $\log_2 N$-scaling is due to the P2P approach. This compares favourably to the linear scaling of N that we would obtain for a fixed set of servers. The factor of $1/M$ is due to splitting the file into parts.

Theorem 1 *In the simultaneous send/receive model the minimum number of rounds required to complete the dissemination of all file parts is $M + \lfloor \log_2 N \rfloor$. If each round takes $1/M$ units of time, then the minimal makespan is, for all M and N,*

$$T^* = 1 + \frac{\lfloor \log_2 N \rfloor}{M} . \tag{1}$$

Proof. Suppose that $N = 2^n - 1 + x$, for $x = 1, \dots, 2^n$. So $n = \lfloor \log_2 N \rfloor$. The fact that $M+n$ is a lower bound on the number of rounds is straightforwardly seen as follows. There are M different file parts and the server can only upload one file part in each round (or, if network coding is used, one linear combination of the information in the file parts). Thus, it takes at least M rounds until the server has completed sufficient uploading that the whole file can be recovered by one peer. The content of the last of these M uploads contains information that is essential to recovering the file, but this information is now known to only the server and one peer. It must takes at least n further rounds for this information to reach the other $N-1$ peers.

Now we show how the bound can be achieved. The result is trivial for $M = 1$. It is instructive to consider the case $M = 2$ explicitly. If $n = 0$ then $N = 1$ and the result is trivial. If $n = 1$ then N is 2 or 3. Suppose $N = 3$. In the following diagram each line corresponds to a round; each column to a peer. The entries denote the file part that the peer downloads that round. The bold entries indicate downloads from the server; un-bold entries indicate downloads from a peer who has the corresponding part.

$$\begin{array}{ccc}
\mathbf{1} & & \\
 & \mathbf{2} & 1 \\
2 & 1 & \mathbf{2}
\end{array}$$

Thus, dissemination of the two file parts to the 3 users can be completed in 3 rounds. The case $N = 2$ is even easier.

If $n \geq 2$, then in rounds 2 to n each user uploads his part to a peer who has no file part and the server uploads part 2 to a peer who has no file part. We reach a point, shown below, at which a set of 2^{n-1} peers have file part 1, a set of $2^{n-1} - 1$ peers have file part 2, and a set of x peers have no file part (those denoted by $* \cdots *$). Let us call these three sets A_1, A_2 and A_0, respectively.

$$\begin{array}{cccccccccccccccccccccc}
\mathbf{1} \\
 & \mathbf{2} & 1 \\
 & & & \mathbf{2} & 1 & 2 & 1 \\
 & & & & & & & \mathbf{2} & 1 & 2 & 1 & 2 & 1 & 2 & 1 \\
 & & & & & & & & & & & & & & & \vdots \\
 & & & & & & & & & & & & & & & & \mathbf{2} & 1 & \cdots & 2 & 1 & * \cdots *
\end{array}$$

We describe what happens during round $n + 1$, taking cases $x = 1$ and $x > 1$ separately. If $x = 1$, then peers in A_1 upload part 1 to $2^{n-1} - 1$ peers in A_2 and to one peer in A_0. Peers in A_2 upload part 2 to $2^{n-1} - 1$ peers in A_1. The server uploads part 2 to one member of A_1. If $x > 1$, then peers in A_1 upload part 1 to $2^{n-1} - \lfloor x/2 \rfloor$ peers in A_2 and to $\lfloor x/2 \rfloor$ peers in A_0. Peers in A_2 upload part 2 to $2^{n-1} - \lceil x/2 \rceil$ peers in A_1 and to another $\lceil x/2 \rceil - 1$ peers in A_0. The server uploads part 2 to a member of A_0. Thus, at the end of this round $2^n - x$ peers have both file parts, x peers have only file part 1, and $x - 1$ peers have only file part 2. One more round (round $n + 2$) is clearly sufficient to complete the dissemination.

Now consider $M \geq 3$. The server uploads part 1 to one peer in round 1. In rounds $j = 2, \ldots, \min\{n, M - 1\}$, each peer who has a file part uploads his part to another peer who has no file part and the server uploads part j to a peer who has no file part. If $M \leq n$, then in rounds M to n each peer uploads his part to a peer who has no file part and the server uploads part M to a peer who has no file part. As above, we illustrate this with a diagram. Here we show the first n rounds in the case $M \leq n$.

$$\begin{array}{cccccccccccccccccccccc}
\mathbf{1} \\
 & \mathbf{2} & 1 \\
 & & & \mathbf{3} & 1 & 2 & 1 \\
 & & & & & & & \mathbf{4} & 1 & 2 & 1 & 3 & 1 & 2 & 1 \\
 & & & & & & & & & & & & & & & \vdots \\
 & & & & & & & & & & & & & & & & \mathbf{M} & 1 & \cdots & 2 & 1 \\
 & & & & & & & & & & & & & & & & & & & & \vdots \\
 & & & & & & & & & & & & & & & & & & & & & \mathbf{M} \; 1 \cdots 2 \; 1 \; * \cdots *
\end{array}$$

Part	Numbers of the file parts at the ends of rounds					
	n	$n+1$	$n+2$	$n+3$	$\cdots$	$n+M-1$
1	2^{n-1}	2^n	N	N	$\cdots$	N
2	2^{n-2}	2^{n-1}	2^n	N	$\cdots$	N
3	2^{n-3}	2^{n-2}	2^{n-1}	2^n	$\cdots$	N
4	2^{n-4}	2^{n-3}	2^{n-2}	2^{n-1}	$\cdots$	N
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
$M-2$	2^{n-M+2}	2^{n-M+3}	2^{n-M+4}	2^{n-M+5}	$\cdots$	N
$M-1$	2^{n-M+1}	2^{n-M+2}	2^{n-M+3}	2^{n-M+4}	$\cdots$	2^n
M	$2^{n-M+1}-1$	$2^{n-M+2}-1$	$2^{n-M+3}-1$	$2^{n-M+4}-1$	$\cdots$	2^n-1

Table 1: Number of file part replica as obtained with our algorithm.

set	peers in the set have	number of peers in set
B_{12}	parts 1 and 2	$2^{n-1} - \lfloor x/2 \rfloor$
B_{1p}	part 1 and a part other than 1 or 2	$2^{n-1} - \lceil x/2 \rceil$
B_1	just part 1	x
B_2	just part 2	$\lfloor x/2 \rfloor$
B_p	just a part other than 1 or 2	$\lceil x/2 \rceil - 1$

Table 2: File parts held by various sets of peers at the end of round $n+1$.

When round n ends, $2^n - 1$ peers have one file part and x peers have no file part. The number of peers having file part i is given in the second column of Table 1. Our algorithm works for any M and N. To see what happens if $M > N$ (and indeed if $M > \lceil \log_2 N \rceil$) one should read as 0 any entry in the table which evaluates to less than 1; for example, the bottom two entries in column 2 and the bottom entry in column 3 are 0 for $n = M - 2$. Now in round $n+1$, by downloading from every peer who has a file part, and downloading part $\min\{n+1, M\}$ from the server, we can obtain the numbers shown in the third column. Moreover, we can easily arrange so that peers can be divided into the sets B_{12}, B_{1p}, B_1, B_2 and B_p as shown in Table 2. In round $n+2$, $x-1$ of the peers in B_1 upload part 1 to peers in B_2 and B_p. Peers in B_{12} and B_2 each upload part 2 to the peers in B_{1p} and to $\lceil x/2 \rceil$ of the peers in B_1. The server and the peers in B_{1p} and B_p each upload a part other than 1 or 2 to the peers in B_{12} and to the other $\lfloor x/2 \rfloor$ peers in B_1. The server uploads part $\min\{n+2, M\}$ and so we obtain the numbers in the fourth column of Table 1. Now all peers have part 1 and so it can be disregarded subsequently. Moreover, we can make the downloads from the server, B_{1p} and B_p so that (disregarding part 1) the number of peers who ultimately have only part 3 is $\lfloor x/2 \rfloor$. This is possible because the size of B_p is no more than $\lfloor x/2 \rfloor$; so if j peers in B_p have part 3 then we can upload part 3 to exactly $\lfloor x/2 \rfloor - j$ peers in B_1. Thus, a similar partitioning into sets as in Table 2 will hold as we start step $n+3$ (when parts 2 and 3 takes over the roles of parts 1 and 2 respectively).

We continue similarly in subsequent rounds, until at the end of round $n+M-1$, all peers have parts $1, \dots, M-2$, $2^n - x$ peers also have both part $M-1$ and part M, x peers also have only part $M-1$, and $x-1$ peers also have only part M. It now takes just a final round to ensure that all peers have parts $M-1$ and M. ■

Note that the optimal strategy above follows two principles. As many different peers as possible obtain file parts early on so that they can start to assist in uploading, and the maximal possible upload capacity is used. Moreover, there is a certain balance in the upload of different file parts so that no part gets circulated too late.

It is interesting that not *all* the available upload capacity is used. Suppose $M \geq 2$. Observe that in round k, for each $k = n+2, \dots, n+M-1$, only $x-1$ of the x peers (in set B_1) who have only file part $k-n-1$ make an upload. This happens $M-2$ times. Also, in round $n+M$ there are only $2x-1$ uploads, whereas $N+1$ are possible. Overall, we use $N+M-2x$ less uploads than we might. It can be checked that this number is the same for $M=1$.

Suppose we were to follow a schedule that uses only x uploads during round $n+1$, when the last peer gets its first file part. We would be using $2^n - x$ less uploads than we might in this round. Since $2^n - x \leq N+M-2x$, we see that the schedule used in the proof above wastes at least as many uploads. So the mathematically interesting question arises as to whether or not it is necessary to use more than x uploads in round $n+1$. In fact, $(N+M-2x)-(2^n-x) = M-1$, so, in terms of the total number of uploads, such a scheduling could still afford not to use one upload during each of the last $M-1$ rounds. The question is whether or not each file part can be made available sufficiently often.

The following example shows that if we are not to use more than x uploads in round $n+1$ we will have to do something quite subtle. We cannot simply pick any x out of the 2^n uploads possible and still hope that an optimal schedule will be *shiftable*: by which we mean that the number of copies of part j at the end of round k will be the same as the number of copies of part $j-1$ at the end of round $k-1$. It is the fact that the optimal schedule used in Theorem 1 is shiftable that makes its optimality so easy to see.

Example 1 *Suppose $M = 4$ and $N = 13 = 2^3 + 6 - 1$, so $M + \lfloor \log_2 N \rfloor = 7$. If we follow the same schedule as in Theorem 1, we reach after round 3,*

$$\mathbf{1} \quad \mathbf{2} \quad 1 \quad \mathbf{3} \quad 1 \quad 2 \quad 1 \quad . \quad . \quad . \quad . \quad . \quad .$$

Now if we only make $x = 6$ uploads during round 4, then there are eight ways to choose which six parts to upload and which two parts not to upload. One can check that in no case is it possible to arrange the parts to upload so that once this is done and uploads are made for round 5 then the resulting state has the same numbers of parts 2, 3 and 4, respectively, as the numbers of parts 1, 2 and 3 at the end of round 4. That is, there is no shiftable optimal schedule. In fact, if our six uploads had been four part 1s and two part 2s, then it would not even be possible to achieve (1).

In some cases, we can achieve (1), if we relax the demand that the schedule be shiftable. Indeed, we conjecture that this is always possible for at least one schedule that uses only x uploads during round $n+1$. However, the fact that we cannot use essentially the same strategy in each round makes the general description of a non-shiftable optimal schedule very complicated. Our aim has been to find an optimal (shiftable) schedule that is easy to describe. We have shown that this is possible if we do use the spare capacity at round $n+1$. For practical purposes this is desirable anyway, since even if it does not affect the makespan it is better if users obtain file parts earlier.

When $x = 2^n$ our schedule can be realized using matchings between the 2^n peers holding the part that is to be completed next and the server together with the $2^n - 1$ peers holding the remaining parts. Otherwise this is not always possible to schedule only with matchings. This is why our solution would not work for the more constrained bidirectional telephone model considered in [2] (where, in fact, the answer differs as N is even or odd).

The solution of the simultaneous send/receive broadcasting model problem now gives the solution of our original uplink-sharing model when all capacities are the same.

Theorem 2 *In the uplink sharing model the minimum makespan is the same as in the simultaneous send/receive model, namely, given by* (1) *for all M and N.*

Proof. Lemmas 1 and 2 show that for the uplink-sharing model with all upload capacities equal to 1 there is an optimal schedule in which at all times each peer is uploading at most one file part to just one other peer. Thus, by the same argument as in the first paragraph of the proof of Theorem 1, we have that (1) is a lower bound on the makespan. Theorem 1 shows that this lower bound can be attained. ■

In fact, the proof of Theorem 1 shows that there is an optimal schedule in which no peer downloads more than a single file part at a time. Thus, we also have the following result.

Theorem 3 *In the uplink-sharing model with all upload capacities equal to 1, constraining the peers' download rates to 1 does not further increase the minimal makespan.*

4 Centralized Solution for General Capacities

We now consider the optimal centralized solution in the general case of the uplink-sharing model in which the upload capacities may be different. Essentially, we have an unusual type of precedence-constrained job scheduling problem. In Section 4.1 we formulate it as a mixed integer linear program (MILP). The MILP can also be used to find approximate solutions of bounded size of sub-optimality. In practice, it is suitable for a small number of file parts M. We discuss its implementation in Section 4.2. Finally, we provide additional insight into the solution with different capacities by considering special choices for N and M when $C_1 = C_2 = \cdots = C_N$, but C_S might be different (Sections 4.3 and 4.4).

4.1 MILP formulation

In order to give the MILP formulation, we need the following lemma. Essentially, it shows that time can be discretized suitably.

Lemma 3 *Consider the uplink-sharing model and suppose all uplink capacities are integer multiples of a common time unit. Then there exists τ, such that under an optimal schedule all uploads start and finish at integer multiples of τ.*

Proof. Without loss of generality, let us suppose that $C_S, C_1, \ldots, C_N$ are all positive integers. Let L be their least common multiple. The time that the first job completes must be an integer multiple of $1/ML$. All remaining jobs are of sizes $1/M$ or $1/M - (1/MC_j)C_i$ for various $C_i \leq C_j$. These are also integer multiples of $1/ML$. Repeating this, we find that the time that the second job completes, and the lengths of all remaining jobs at this point must be integer multiples of $1/(ML)^2$. Repeating further, we find that $\tau = 1/(ML)^{MN}$ suffices. ■

We next show how the solution to the general problem can be found by solving a number of linear programs. Let 'slot t' be the interval of time $[(t-1)\tau, t\tau)$, $t = 0, \ldots$. Recall that the uploads of all file parts start and finish at the ends of such slots. We let the server be denoted with index 0. Let $x_{ijk}(t)$ be 1 or 0 as peer i does or does not download file part k from peer j during slot t. Let $p_{ik}(t)$ denote the proportion of file part k that peer i has downloaded by the end of slot t. Since slots are of length τ, the minimum time needed to disseminate all file parts is the least value of $T\tau$ for which all file parts can be uploaded within T slots. T corresponds to the least integer S for which the optimal value of the following MILP is the total number of file parts, MN. Since this T is certainly greater than $1/(C_S\tau)$ and less than $N/(C_S\tau)$, we can search for its value by a simple bisection search, solving the MILP with objective function

$$\text{maximize} \quad \sum_{i,k} p_{ik}(S), \tag{2}$$

for various integers S, subject to the constraints given below. The source availability constraint (6) guarantees that a user has completely downloaded a part before he can upload it to his peers (here $\xi_{jk}(t) = \mathbf{1}\{p_{jk}(t) = 1\}$). The connection constraint (7) requires that each user only carries out a single upload at a time. This is justified by Lemma 1 which also saves us another essential constraint and variable to control the actual download rates. The single user downloading from peer j at time t will do so at rate C_j as expressed in the link constraint (5). Continuity and stopping constraints (8, 9) require that a download that has started will not be interrupted until completion and then be stopped. The exclusivity constraint (10) ensures that each user downloads a given file part from exactly one other user. Stopping and exclusivity constraints are not based on assumptions, but obvious constraints to exclude redundant uploads.

Regional constraints

$$x_{ijk}(t) \in \{0, 1\} \text{ for all } i, j, k, t \tag{3}$$

$$p_{ik}(t) \in [0, 1] \text{ for all } i, k, t \tag{4}$$

Link constraints between variables

$$p_{ik}(t) = M\tau \sum_{t'=1}^{t} \sum_{j=0}^{N} x_{ijk}(t')C_j \text{ for all } i, k, t \tag{5}$$

Essential constraints

$$x_{ijk}(t) - \xi_{jk}(t) \leq 0 \text{ for all } i, j, k, t \quad \text{(Source availability constraint)} \tag{6}$$

$$\sum_{i,k} x_{ijk}(t) \leq 1 \text{ for all } j, t \quad \text{(Connection constraint)} \tag{7}$$

$$x_{ijk}(t) - \xi_{ik}(t+1) - x_{ijk}(t+1) \leq 0 \text{ for all } i, j, k, t \quad \text{(Continuity constraint)} \tag{8}$$

$$x_{ijk}(t) + \xi_{ik}(t) \leq 1 \text{ for all } i, j, k, t \quad \text{(Stopping constraint)} \tag{9}$$

$$\sum_{j} x_{ijk}(t) \leq 1 \text{ for all } i, k, t \quad \text{(Exclusivity constraint)} \tag{10}$$

Initial conditions

$$p_{0k}(0) = 1 \text{ for all } k \tag{11}$$

$$p_{ik}(0) = 0 \text{ for all } i, k \tag{12}$$

Linearization constraints

$$\xi_{ik}(t) \in \{0, 1\} \text{ for all } \ i, k, t \tag{13}$$

$$p_{ik}(t) - \xi_{ik}(t) \geq 0 \text{ and } p_{ik}(t) - \xi_{ik}(t) < 1 \ \text{ for all } i, k, t \tag{14}$$

4.2 Implementation of the MILP

MILPs are well-understood and there exist efficient computational methods and program codes. However, as the numbers of variables and constraints in the LP grows exponentially in N and M, this approach is not practical for large N and M.

Nonetheless, we can use the MILP to bound the minimum makepan by replacing τ with a larger value. Suppose τ is such that the conclusion of Lemma 3 holds. Pick τ', with $\tau' > \tau$. Consider the MILP with τ' replacing τ, and with "$\leq$" replacing "$=$" in (5) to accommodate the fact that a job can now complete part way through a slot. If T' is the least integer S such that the optimal value of the MILP is NM then $T'\tau'$ is the minimum makespan in a problem in which there is the requirement that jobs may start only at times that are integer multiples of τ' and in which the makespan is taken to be the first time which is an integer multiple of τ' and all jobs are complete, so $T'\tau' = T\tau$. Now consider an optimal schedule for the original problem. Let the NM jobs be indexed in the order of their start times in this schedule. Suppose job j is the first such indexed job whose start time is not an integer multiple of τ'. Let us delay the schedule from this point onward by increasing the start times of jobs $j, \dots, NM$ by the same amount s, with $s < \tau'$, so that j now starts at an integer multiple of τ'. We repeat this process on the resulting schedule until we finally have a schedule in which all jobs start at times that are integer multiples of τ'. This produces a feasible schedule for the modified problem. It is clear that no start time has increased by more that $NM\tau'$, and so this is also a bound on the increase in makespan. Thus, we may conclude that $T'\tau' \leq T\tau + NM\tau'$. So $T'\tau' - NM\tau'$ and $T'\tau'$ are lower and upper bounds on $T\tau$, the minimal makespan in the original problem, and are easier to compute than $T\tau$.

4.3 Insight for special cases with small N and M

We now provide some insight into the minimal makespan solution with different capacities by considering special choices for N and M when $C_1 = C_2 = \cdots = C_N$, but C_S might be different. This addresses the case of the server having a significantly higher upload capacity than the end users.

Suppose $N = 2$ and $M = 1$, that is, the file has not been split. Only the server has the file initially, thus either (a) both peers download from the server, in which case the makespan is $T = 2/C_S$, or (b) one peer downloads from the server and then the second peer downloads from the first; in this case $T = 1/C_S + 1/C_1$. Thus, the minimal makespan is $T^* = 1/C_S + \min\{1/C_S, 1/C_1\}$.

If $N = M = 2$ we can again adopt a brute force approach. There are 16 possible cases, each specifying the download source that each peer uses for each part. These can be reduced to four by symmetry.

Case A: Everything is downloaded from the server. This is effectively the same as case (a) above. When C_1 is small compared to C_S, this is the optimal strategy.
Case B: One peer downloads everything from the server. The second peer downloads from the first. This is as case (b) above, but since the file is split in two, T is less.
Case C: One peer downloads from the server. The other peer downloads one part of the file from the server and the other part from the first peer.
Case D: Each peer downloads exactly one part from the server and the other part from the other peer. When C_1 is large compared to C_S, this is the optimal strategy.

In each case, we can find the optimal schedule and hence the minimal makespan. This is shown in Table 3.

case	makespan
A	$\frac{2}{C_S}$
B	$\frac{1}{2C_S} + \frac{1}{2C_1} + \max\left(\frac{1}{2C_S}, \frac{1}{2C_1}\right)$
C	$\frac{1}{2C_S} + \max\left(\frac{1}{C_S}, \frac{1}{2C_1}\right)$
D	$\frac{1}{C_S} + \frac{1}{2C_1}$

Table 3: Minimal makespan in the four possible cases when $N = M = 2$.

The optimal strategy arises from A, C or D as C_1/C_S lies in the intervals $[0, 1/3]$, $[1/3, 1]$ or $[1, \infty)$ respectively. In $[1, \infty)$, B and D yield the same makespan. See Figure 1. Note that under the optimal schedule for case C one peer has to wait while the other starts downloading. This illustrates that greedy-type distributed algorithms may not be optimal and that restricting uploaders to a single upload is sometimes necessary for an optimal schedule (cf. Section 2).

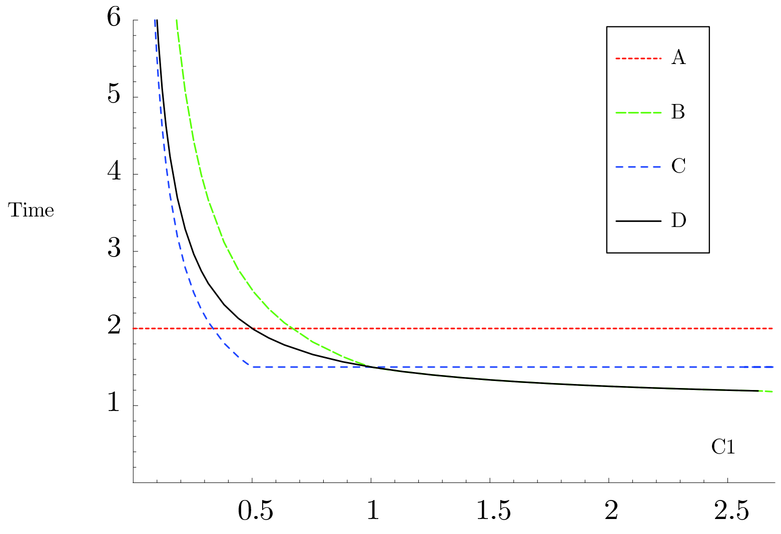


Figure 1: Minimal makespan as a function of C_1/C_S in the four possible cases when $N = M = 2$.

4.4 Insight for special cases with large M

We still assume $C_1 = C_2 = \cdots = C_N$, but C_S might be different. In the limiting case that the file can be divided into infinitely many parts, the problem can be easily solved for any number N of users. Let each user download a fraction $1-\alpha$ directly from the server at rate C_S/N and a fraction $\alpha/(N-1)$ from each of the other $N-1$ peers, at rate $\min\{C_S/N, C_1/(N-1)\}$ from each. The makespan is minimized by choosing α such that the times for these two downloads are equal, if possible. Equating them, we find the minimal makespan as follows.

Case 1: $C_1/(N-1) \le C_S/N$:

$$\frac{(1-\alpha)N}{C_S} = \frac{\alpha}{C_1} \quad \Longrightarrow \quad \alpha = \frac{NC_1}{C_S + NC_1} \quad \Longrightarrow \quad T = \frac{N}{C_S + NC_1}\,. \tag{15}$$

Case 2: $C_1/(N-1) \ge C_S/N$:

$$\frac{(1-\alpha)N}{C_S} = \frac{\alpha N}{(N-1)C_S} \quad \Longrightarrow \quad \alpha = \frac{N-1}{N} \quad \Longrightarrow \quad T = \frac{1}{C_S}\,. \tag{16}$$

In total, there are N MB to upload and the total available upload capacity is $C_S + NC_1$ MBps. Thus, a lower bound on the makespan is $N/(C_S + NC_1)$ seconds. Moreover, the server has to upload his file to at least one other user. Hence another lower bound on the makespan is $1/C_S$. The former bound dominates in case 1 and we have shown that it can be achieved. The latter bound dominates in case 2 and we have shown that it can be achieved. As a result, the minimal makespan is

$$T^* = \max\left\{\frac{1}{C_S}, \frac{N}{C_S + NC_1}\right\}\,. \tag{17}$$

Figure 2 shows the minimal makespan when the file is split in 1, 2 and infinitely many file parts when $N = 2$. It illustrates how the makespan decreases with M.

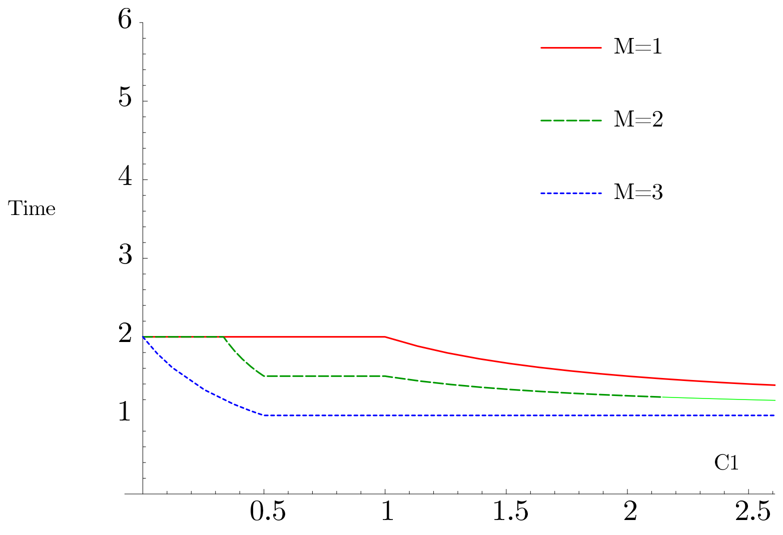


Figure 2: Minimal makespan as a function of C_1/C_S for different values of M when $N = 2$.

In the next section, we extend the results in this limiting case to a much more general scenario.

5 Centralized Fluid Limit Solution

In this section, we generalize the results of Section 4.4 to allow for general capacities C_i. Moreover, instead of limiting the number of sources to one designated server with a file to disseminate, we now allow every user i to have a file that is to be disseminated to all other users. We provide the centralized solution in the limiting case that the file can be divided into infinitely many parts.

Let $F_i \geq 0$ denote the size of the file that user i is to disseminate to all other users. Seeing that in this situation there is no longer one particular server and everything is symmetric, we change notation for the rest of this section so that there are $N \geq 2$ users $1, 2, \ldots, N$. Moreover, let $F = \sum_{i=1}^{N} F_i$ and $C = \sum_{i=1}^{N} C_i$. We will prove the following result.

Theorem 4 *In the fluid limit, the minimal makespan is*

$$T^* = \max\left\{\frac{F_1}{C_1}, \frac{F_2}{C_2}, \ldots, \frac{F_N}{C_N}, \frac{(N-1)F}{C}\right\} \tag{18}$$

and this can be achieved with a two-hop strategy, i.e., one in which users i's file is uploaded to user j, either directly from user i, or via at most one intermediate user.

Proof. If $N = 2$ then the minimal makespan is clearly $\max\{F_1/C_1, F_2/C_2\}$. This is the value of T^* in (18).

Now consider $N \geq 3$. Each user has to upload his file at least to one user, which takes time F_i/C_i. Moreover, the total volume of files to be uploaded is $(N-1)F$ and the total available capacity is C. Thus, the makespan is at least T^*. Thus, each of the $N+1$ terms within the braces on the right hand side of (18) are lower bounds on the makespan. It remains to show that a makespan of T^* can be achieved. There are two cases to consider.

Case 1: $(N-1)F/C \geq \max_i F_i/C_i$ for all i and $T^* = (N-1)F/C$.

Let us adopt a protocol in which peer i sends a portion $\alpha_i F_i$ of its file directly to each of the other $N-1$ peers, and sends a nonoverlapping portion $\alpha_j F_i$ of its file to peer j, who is then responsible for forwarding it to each of the remaining $(N-2)$ peers. Let

$$\alpha_i = \frac{(N-1)C_i}{(N-2)C} - \frac{F_i}{(N-2)F}\,.$$

Note that $\sum_i \alpha_i = 1$, and that by the assumptions of this case $\alpha_i \geq 0$.

Peer i now makes these uploads at constant rates, sending its own file to peer j at rate $(\alpha_i + \alpha_j)F_i/T^*$ and forwarding file k to other peers at rate $(N-2)\alpha_i F_k/T^*$. The sum of these rates is

$$\frac{(N-1)\alpha_i F_i + (1-\alpha_i)F_i + (N-2)\alpha_i(F - F_i)}{T} = C_i\,,$$

and so peer i is indeed uploading at maximum rate. Notice that peer i receives the portion of file k that it must forward at rate $\alpha_i F_k/T^*$, which is fast enough for it to maintain the forwarding rate of $(N-2)\alpha_i F_k/T^*$ to the other peers.

Case 2: $F_i/C_i > (N-1)F/C$, and $T^* = F_i/C_i$, for some i.

Peer i now spends all the time T^* uploading its file to other peers. Suppose we reduce the capacity of all other peers according to the formula

$$C'_j = \frac{C_i}{F_i} \frac{(N-2)FF_iC_j + F_jF_1C - (N-1)FF_jC_i}{F_iC - FC_i}, \quad j \neq i\,. \tag{19}$$

It is not hard to check that $C'_j \leq C_j$ and $0 \leq F_j/C'_j \leq F_i/C_i$. With $C' = C_i + \sum_{j\neq i} C'_j$ we also have $(N-1)F/C = F_i/C_i$. We have reduced the problem to a case to which we can apply the protocol of Case 1. ■

In practice, the file will not be infinitely divisible. However, we often have $M >> \log(N)$ and this appears to be sufficient for (18) to be a good approximation. Thus, the fluid limit approach of this section is suitable for typical and for large values of M.

6 Decentralized Solution for Equal Capacities

In order to give a lower bound on the minimal makespan, we have been assuming a centralized controller does the scheduling. We now consider a naive randomized strategy and investigate the loss in performance that is due to the lack of centralized control. We do this for equal capacities and in two different information scenarios, evaluating its performance by analytic bounds, simulation as well as direct computation. In Section 6.1 we consider the special case of one file part, in Section 6.2 we consider the general case of M file parts. We find that even this naive strategy disseminates the file in an expected time whose growth rate with N is similar to the growth rate of the minimal time that we have found for a centralized controller (cf. Section 3). This suggests that the performance of necessarily decentralized P2P file dissemination systems should still be close to our performance bounds so that they are useful in practice.

6.1 The special case of one file part

Assumptions

Let us start with the case $M = 1$. We must first specify what information is available to users. It makes sense to assume that each peer knows the number of parts into which the file is divided, M, and the address of the server. However, a peer might not know N, the total number of peers, nor its peers' addresses, nor if they have the file, nor whether they are at present occupied uploading to someone else.

We consider two different information scenarios. In the first one, *List*, the number of peers holding the file and their addresses are known. In the second one, *NoList*, the number and addresses of all peers are known, but not which of them currently hold the file. Thus, in *List*, downloading users choose uniformly at random between the server and the peers already having the file. In *NoList*, downloading users choose uniformly amongst the server and all their peers. If a peer receives a query from a single peer, he uploads the file to that peer. If a peer receives queries from multiple peers, he chooses one of them uniformly at random. The others remain unsuccessful in that round. Thus, in *List* transmission can fail only if too many users try to download simultaneously from the same uploader. In *NoList*, transmission might also fail if a user tries to download from a peer who does not yet have the file.

Theoretical Bounds

The following theorem explains how the expected makespan that is achieved by the randomized strategy grows with N, in both the *List* and the *NoList* scenarios.

Theorem 5 *In the uplink-sharing model, with equal upload capacities, the expected number of rounds required to disseminate a single file to all peers in either the List or NoList scenario is* $\Theta(\log N)$.

Proof. By Theorem 1 that the running time is $\Omega(\log N)$. So we need only show that it is also $O(\log N)$. In the *List* scenario our simple randomized algorithm runs in less time than in the *NoList* scenario. Since already have the lower bound given by Theorem 1, it suffices to prove that the expected running time in the *NoList* scenario is $O(\log N)$. There is also similar direct proof that the expected running time under the *List* scenario is $O(\log N)$.

Suppose we have reached a stage in the dissemination at which n_1 peers (including the server) have the file and n_0 peers do not, with $n_0+n_1 = N+1$. (The base case is $n_1 = 1$, when only the server has the file.) Each of the peers that does not have the file randomly chooses amongst the server and all his peers (*NoList*) and tries to download the file. If more than one peer tries to download from the same place then only one of the downloads is successful. The proof has two steps.

(i) Suppose that $n_1 \leq n_0$. Let i be the server or a peer who has the file and let I_i be an indicator random variable that is 0 or 1 as i does or does not upload it in the next slot. Let $Y = \sum_i I_i$, where the sum is taken over all n_1 peers who have the file. Thus $n_1 - Y$ is the number of uploads that take place. Then

$$EI_i = \left(1 - \frac{1}{N}\right)^{n_0} \leq \left(1 - \frac{1}{2n_0}\right)^{n_0} \leq \frac{1}{\sqrt{e}}\,. \tag{20}$$

Now since $E(\sum_i I_i) = \sum_i EI_i$, we have $EY \leq n_1/\sqrt{e}$. Thus, by the Markov inequality, that for a nonnegative random variable Y we have that for any k (not necessarily an integer) $P(Y \geq k) \leq (1/k)EY$, we have by taking $k = (2/3)n_1$,

$$P\big(n_1 - Y \equiv \text{number of uploads} \leq \tfrac{1}{3}n_1\big) = P(Y \geq \tfrac{2}{3}n_1) \leq \frac{n_1/\sqrt{e}}{\frac{2}{3}n_1} = 3/(2\sqrt{e}) < 1\,. \tag{21}$$

Thus the expected number of steps required for the number of peers who have the file to increases from n_1 to at least $n_1 + (1/3)n_1 = (4/3)n_1$ is bounded by a geometric random variable with mean $\mu = 1/(1 - 3/(2\sqrt{e}))$. This implies that we will reach a state in which more peers have the file than do not in an expected time that is $O(\log N)$. From that point we continue with step (ii) of the proof.

(ii) Suppose $n_1 > n_0$. Let j be a peer who does not have the file and let J_j be an indicator random variable that is 0 or 1 as peer j does or does not succeed in downloading it in the next slot. Let $Z = \sum_j J_j$, where the sum is taken over all n_0 peers who do not have the file. Suppose X is the number of the other $n_0 - 1$ peers that try to download from the same place as does peer j. Then

$$\begin{aligned}
P(J_j = 0) &= E\left[\frac{n_1}{N}\left(\frac{1}{1+X}\right)\right] \\
&\geq E\left[\frac{n_1}{N}(1-X)\right] \\
&= \frac{n_1}{N}\left(1 - \frac{n_0 - 1}{N}\right) \\
&= \frac{n_1}{N}\left(1 - \frac{N - n_1}{N}\right) \\
&= \frac{n_1^2}{N^2} \\
&\geq 1/4 \, . \qquad (22)
\end{aligned}$$

Hence $EZ \leq (3/4)n_0$ and so, again using the Markov inequality,

$$P\big(n_0 - Z \equiv \text{number of downloads} \leq \tfrac{1}{8}n_0\big) = P\big(Z \geq \tfrac{7}{8}n_0\big) \leq \frac{\frac{3}{4}n_0}{\frac{7}{8}n_0} = \tfrac{6}{7} \, . \qquad (23)$$

It follows that the number of peers who do not yet have the file decreases from n_0 to no more than $(7/8)n_0$ in an expected number of steps no more than $\mu' = 1/(1 - \frac{6}{7}) = 7$. Thus the number of steps needed for the number of peers without the file to decrease from n_0 to 0 is $O(\log n_0) = O(\log N)$. In fact, this is a weak upper bound. By more complicated arguments we can show that if $n_0 = aN$, where $a \leq 1/2$, then the expected remaining time for our algorithm to complete under *NoList* is $\Theta(\log\log N)$. For $a > 1/2$ the expected time remains $\Theta(\log N)$. ■

Simulation

For the problem with one server and N users we have carried out 1000 independent simulation runs[4] for a large range of parameters values, $N = 2, 4, \ldots, 2^{25}$. We found that the achieved expected makespan appears to grow as $a + b \times \log_2 N$. Motivated by this and the theoretical bound from Theorem 5 we fitted the linear model

$$y_{ij} = \alpha + \beta x_i + \epsilon_{ij} \, , \qquad (24)$$

where y_{ij} is the makespan for $x_i = \log_2 2^i = i$, obtained in run j, $j = 1, \ldots, 1000$. Indeed, the model fits the data very well in both scenarios. Figure 3 shows the scatter plots. The following results enable us to compare the expected makespan of the naive randomized strategy to the that of a centralized controller.

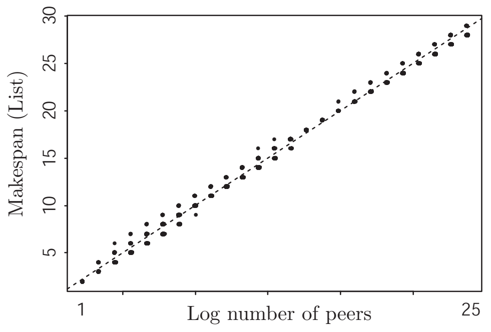
For *List*, the regression provides a good fit, with R-squared value of 0.9975 and p-value 0. The regression line is

$$1.1392 + 1.1021 \times \log_2 N \, . \qquad (25)$$

For *NoList*, there is more variation in the data than for *List*, but, again, a linear regression provides a good fit, with R-squared of 0.9864 and p-value 0. The regression line is

$$1.7561 + 1.5755 \times \log_2 N \, . \qquad (26)$$

[4]As many as 1000 runs were required for the comparison with the computational results in Tables 4 and 5, mainly because the makespan always takes integer values.



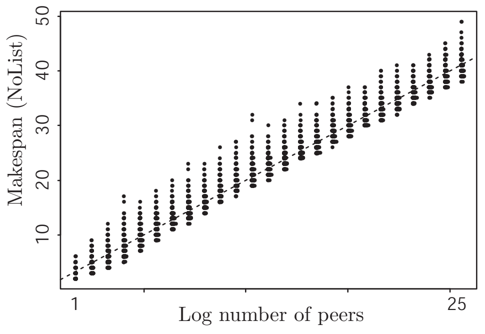


Figure 3: Scatter plots of simulation data for *List* and *NoList*.

As expected, the additional information for *List* leads to a significantly lesser makespan when compared to *NoList*, in particular the log-term coefficient is significantly smaller. In the *List* scenario, the randomized strategy achieves a makespan that is very close to the centralized optimum of $1 + \lfloor \log_2 N \rfloor$ of Section 3: It is only suboptimal by about 10%. Hence even this simple randomized strategy performs well in both cases and very well when state information is available, suggesting that our bounds are useful in practice.

Computations

Alternatively, it is possible to compute the mean makespan analytically by considering a Markov Chain on the state space $0, 1, 2, \ldots, N$, where state i corresponds to i of the N peers having the file. We can calculate the transition probabilities p_{ij}. In the *NoList* case, for example, following the Occupancy Distribution (c.f. [16]), we obtain

$$p_{i,i+m} = \sum_{j=i-m}^{i} \frac{(-1)^{j-i+m} i!}{(i-j)!(i-m)!(j-i+m)!} \left(\frac{N-1-j}{N-1} \right)^{N-i} . \tag{27}$$

Hence, letting $k(i)$ denote the expected time to hit state N from i, we have

$$k(i) = \frac{1 + \sum_{j>i} k(j) p_{i,j}}{1 - p_{i,i}} . \tag{28}$$

The quantity of interest is $k(0)$. Although in principle, $k(0)$ can be computed from (27) and (28), in practice its accurate computation is very computationally demanding.

For *NoList* (and by similar formulae for *List*) we compute the expected makespans shown in Tables 4 and 5, for $N = 2, 4, \ldots, 2^9$. We also find the expected makespans from simulation. The computed and simulated values differ by only small amounts, without any apparent trend. It can also be checked by computing the standard deviation that the computed mean makespan is contained in the approximate 95% confidence interval of the simulated mean makespan. The only exception is for $N = 128$ for *NoList* where it is just outside by approximately 0.0016. The computations support the accuracy of our simulation results. However, exact computations are computationally too difficult for larger N. Even at the small value of $N = 2^9 = 512$ we must be careful if we are to get correct answers. We compute as rationals the 130,816 probabilities $p_{i,i+m}$. We then carry out a similar number of multiplications and additions in double precision arithmetic to find $k(0)$ from (28).

N	sim.	comp.	difference
2	2.000	2.000	$= 0.000$
4	3.089	3.083	$+0.006$
8	4.167	4.172	-0.005
16	5.333	5.319	$+0.014$
32	6.534	6.538	-0.004
64	7.806	7.794	$+0.012$
128	8.994	8.981	$+0.013$
256	10.059	10.057	$+0.002$
512	11.107	11.116	-0.009

Table 4: Simulated and computed mean makespans for *List* are close.

N	sim.	comp.	difference
2	2.314	2.333	-0.019
4	4.071	4.058	$+0.013$
8	5.933	5.956	-0.023
16	7.847	7.867	-0.020
32	9.689	9.710	-0.021
64	11.430	11.475	-0.045
128	13.092	13.173	-0.081
256	14.827	14.819	$+0.008$
512	16.426	16.427	-0.001

Table 5: Simulated and computed mean makespans for *NoList* are close.

6.2 The general case of M file parts

Assumptions

We now consider splitting the file into several file parts. With the same assumptions as in the previous section, we repeat the analysis for *List* for various values of M. Thus, in each round, a downloading user connects to a peer chosen uniformly at random from those peers that have at least one file part that the user does not yet have. An uploading peer randomly chooses one out of the peers requesting a download from him. He uploads to that peer a file part that is randomly chosen from amongst those that he has and the peer still needs.

Simulation

In is too computationally demanding to carry out the sort of exact calculations we performed in the case of a single file part, so we use only simulation to estimate the makespan. Again, we consider a large range of parameter. We carried out 100 independent runs for each $N = 2$, $4, \ldots, 2^{15}$. For each value of $M = 1 - 5$, 8, 10, 15, 20, 50 we fitted the linear model (24).

Table 6 summarizes the simulation results. The R-squared values indicate a good fit, although the fact that these decrease with M suggests there may be a finer dependence on M or N. The final column of the table can be compared with the coefficient of $\log_2 N$ in the second column (remembering that Theorem 1 gave a lower bound on makespan of order $(1/M)\log_2 N$)). In fact, we obtain a better fit using Generalized Additive Models (cf. [12]). However, our interest here is not in fitting the best possible model, but to compare the growth rate with N to the one obtained in the centralized case in Section 3. Moreover, from the diagnostic plots we note that the actual performance for large N is better than given by the regression line, increasingly so for increasing M. In each case, we obtain significant p-values. The regression $0.7856 + 1.1520 \times \log_2 N$ for $M = 1$ does not quite agree with $1.1392 + 1.1021 \times \log_2 N$ found in (25).

We conclude that, as in the centralized scenario, the makespan can also be reduced significantly in a decentralized scenario even when a simple randomized strategy is used to disseminate the file parts. However, as we note by comparing the second and fourth columns of Table 6, as M increases the achieved makespan compares less well relative to the centralized minimum of $1 + (1/M)\lfloor \log_2 N \rfloor$. In particular, note the slower decrease of the log-term coefficient. This is depicted in Figure 5.

M	Fitted makespan	R-squared	$1/M$
1	$0.7856 + 1.1520 \times \log_2 N$	0.9947	1.000
2	$1.3337 + 0.6342 \times \log_2 N$	0.9847	0.500
3	$1.4492 + 0.4561 \times \log_2 N$	0.9719	0.333
4	$1.4514 + 0.3661 \times \log_2 N$	0.9676	0.250
5	$1.4812 + 0.3045 \times \log_2 N$	0.9690	0.200
8	$1.4907 + 0.2113 \times \log_2 N$	0.9628	0.125
10	$1.4835 + 0.1791 \times \log_2 N$	0.9602	0.100
15	$1.4779 + 0.1326 \times \log_2 N$	0.9530	0.067
20	$1.4889 + 0.1062 \times \log_2 N$	0.9449	0.050
50	$1.4524 + 0.0608 \times \log_2 N$	0.8913	0.020

Table 6: Simulation results in the decentralized *List* scenario for various values of M and log-term coefficients in the centralized optimum (cf. Theorem 1).

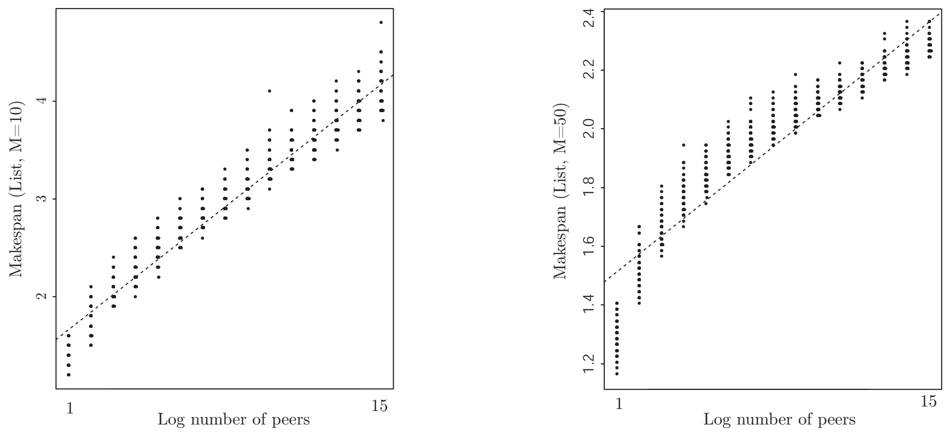


Figure 4: Scatter plots of data for *List*, with $M = 10$ and $M = 50$.

Still, we have seen that even this naive randomized strategy disseminates the file in an expected time whose growth rate with N is similar to the growth rate of the minimal time that we have found for a centralized controller in Section 3, confirming our performance bounds are useful in practice. This is confirmed also by initial results of current work on the performance evaluation of the Bullet' system [18].

The program code for simulations as well as the computations and the diagnostic plots used in this section are available on request and will be made available via the Internet[5].

7 Discussion

In this paper, we have given three complementary solutions for the minimal time to fully disseminate a file of M parts from a server to N end users in a centralized scenario, thereby

[5]http://www.statslab.cam.ac.uk/~jm288/

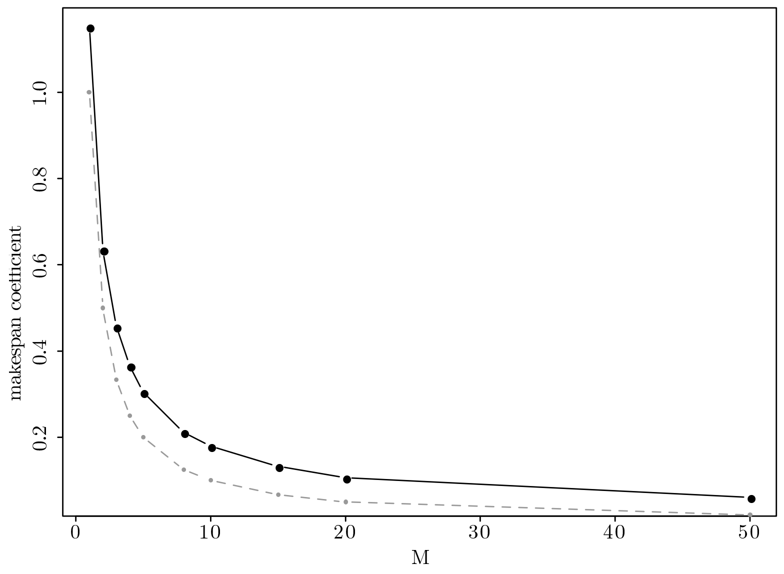


Figure 5: Illustration of the log-term coefficients of the makespan from Table 6: the decentralized *List* scenario (solid) and the idealized centralized scenario (dashed).

providing a lower bound on and a performance benchmark for P2P file dissemination systems. Our results illustrate that the P2P approach, combined with a splitting of the file into M parts, can achieve a significant reduction in makespan. Moreover, the workload of the server is less than in the traditional client/server approach in which it does all the uploading. We have also investigated the loss in efficiency that occurs when no centralized control is pratical. We find that the minum makespan achievable ib necessarily decentralized P2P file dissemination systems can be close to our performance lower bound, thereby confirming their practical use.

It would be interesting to compare dissemination times of various efficient real overlay networks directly to our performance lower bound. A mathematical analysis of the protocols is rarely tractable, but simulation or measurements such as in [15] and [26] for the BitTorrent protocol can be carried out in an environment suitable for this comparison. See also testbed results for Slurpie [29] and simulation results for Avalanche [10]. Our bounds can be compared to the makespan obtained by Bullet' [18]. Initial results confirm their practical use further.

In practice, splitting the file and passing on extra information has an overhead cost. Moreover, when using Transmission Control Protocol (TCP), longer connections are more efficient than shorter ones. TCP is used practically everywhere except for the Internet Control Message Protocol (ICMP) and User Datagram Protocol (UDP) for real-time applications. For further details see [30]. If splitting has an overhead cost then it will not be worthwhile to make M too large. This could be investigated in more detail.

In the proof of Lemma 1 and Lemma 2 we have used fair sharing and continuity assumptions. It would interesting to investigate whether either of these can be relaxed.

It would also be interesting to model to a dynamic setting in which peers join the population of peers randomly. What will happen if peers‘ leave once they have completed their download of the file (a behaviour sometimes called *easy-riding*)? In Internet applications users often connect for only relatively short times. Work in this direction is pursued in [27], using a fluid model to study the steady-state performance, and there is other relevant work in [32]. Also of interest would be to model users who attempt to free-ride by not contributing any uploading effort. The BitTorrent protocol implements a choking algorithm to limit free-riding.

In some scenarios it might be appropriate to assume that users *push* messages rather than *pull* them. See [9] for an investigation of the design space for distributed information systems. The push-pull distinction is also part of their classification. In a push system, the centralized case would remain the same. However, we expect the decentralized case to be different. There are a number of other interesting questions which could be investigated in this context. For example, what happens if only a subset of the users is actually interested in the file, but the uploaders do not know which?

It could be interesting to consider more general upload and download constraints. We might suppose that user i can upload at a rate C_i and simultaneously download at rate D_i. Or we might suppose that it takes time t_{ij} for a file to be sent from user i to user j. This produces a transportation network problem in which the link costs are the link delays. The minimum makespan problem can be phrased as a one-to-all shortest path problem if C_j is at least $N+1$. However, the problem is sufficiently different from traditional shortest path problems that greedy algorithms, induction on nodes and dynamic programming are not readily applied. For $M=1$, Prüfer's $(N+1)^{N-1}$ labelled trees [4], together with the obvious $O(N)$ algorithm for the optimal scheduling on a tree, provides a basis for exhaustive search or a branch and bound algorithm.

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. on Information Theory*, 46:1204–1216, 2000.

[2] A. Bar-Noy, S. Kipnis, and B. Schieber. Optimal multiple message broadcasting in telephone-like communication systems. *Discrete Applied Mathematics*, 100:1–15, 2000.

[3] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Some observations on BitTorrent performance. *Performance Evaluation Review*, 33(1):398–399, 2005.

[4] B. Bollobás. *Modern Graph Theory.* Springer, New York, 1998.

[5] C. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Split-Stream: High-bandwidth multicast in cooperative environments. In *19th ACM Symposium on Operating Systems Principles (SOSP'03)*, 2003.

[6] E. J. Cockayne and A. G. Thomason. Optimal multimessage broadcasting in complete graphs. *Utilitas Math.*, 18:181–199, 1980.

[7] B. Cohen. Incentives build robustness in BitTorrent. *Proceedings of the Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA*, 2003.

[8] A. M. Farley. Broadcast time in communication networks. *SIAM Journal on Applied Mathematics*, 39(2):385–390, October 1980.

[9] M. Franklin and S. Zdonik. A framework for scalable dissemination-based systems. *ACM SIGPLAN Notices*, 32(10):94–105, 1997.

[10] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *IEEE INFOCOM 2005*, March 2005.

[11] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, analysis, and modeling of BitTorrent-like systems. In *Proceedings of Internet Measurement Conference 2005 (IMC 2005)*, October 2005.

[12] T. Hastie and R. Tibshirani. *Generalized Additive Models*. Chapman and Hall, London, 1990.

[13] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.

[14] J. Hromkovic, R. Klasing, B. Monien, and R. Peine. *Combinatorial Network Theory (F. Hsu and D.-Z. Du, Eds.)*, chapter Dissemination of Information in Interconnection Networks (Broadcasting and Gossiping), pages 125–212. Kluwer Academic Publishers, 1995.

[15] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. A. Hamra, and L. Garcés-Erice. Dissecting BitTorrent: Five months in a torrent's lifetime. *Passive and Active Measurements 2004*, 2004.

[16] N. L. Johnson, S. Kotz, and A. W. Kemp. *Univariate Discrete Distributions*. Wiley Europe, 1993.

[17] T. Karagiannis, A. Broido, N. Brownlee, k. claffy, and M. Faloutsos. Is P2P dying or just hiding? In *IEEE Globecom 2004 - Global Internet and Next Generation Networks*, November 2004.

[18] D. Kostić, R. Braud, C. Killian, E. Vandekieft, J. W. Anderson, A. C. Snoeren, and A. Vahdat. Maintaining high-bandwidth under dynamic network conditions. In *USENIX 2005 Annual Technical Conference*, 2005.

[19] C.-H. Kwon and K.-Y. Chwa. Multiple message broadcasting in communication networks. *Networks*, 26:253–261, 1995.

[20] L. Massoulié and M. Vojnović. Coupon replication systems. In *ACM Sigmetrics 2005*, June 2005.

[21] J. Mundinger. *Analysis of Peer-to-Peer Systems in Communication Networks*. PhD thesis, Cambridge University, August 2005.

[22] J. Mundinger and R. R. Weber. Efficient content distribution using peer-to-peer technology. In C. Griwodz, T. P. Plagemann, and R. Steinmetz, editors, *Abstracts Collection – Content Distribution Infrastructures. Dagstuhl Seminar Proceedings*, 04201, 2004.

[23] J. Mundinger, R. R. Weber, and G. Weiss. Analysis of peer-to-peer file dissemination. *To appear in Performance Evalution Review, Eighth Workshop on Mathematical Performance Modeling and Analysis (MAMA 2006) Issue*, 2006.

[24] J. Mundinger, R. R. Weber, and G. Weiss. Analysis of peer-to-peer file dissemination amongst users of different upload capacities. *To appear in Performance Evaluation Review, Performance 2005 Issue*, 2006.

[25] A. Parker. The true picture of peer-to-peer filesharing. CacheLogic, 2004. `http://www.cachelogic.com/`.

[26] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips. The Bittorrent P2P file-sharing system: Measurements and analysis. In *4th International Workshop on Peer-to-Peer Systems (IPTPS'05)*, February 2005.

[27] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *Proc. ACM SIGCOMM*, September 2004.

[28] K. K. Ramachandran and B. Sikdar. An analytic framework for modeling peer to peer networks. *IEEE INFOCOM 2005*, 2005.

[29] R. Sherwood, R. Braud, and B. Bhattacharjee. Slurpie: A cooperative bulk data transfer protocol. *IEEE INFOCOM 2004*, 2004.

[30] R. Srikant. *The Mathematics of Internet Congestion Control*. Birkhäuser, 2004.

[31] X. Yang and G. de Veciana. Service capacity of peer to peer networks. *IEEE INFOCOM 2004*, 2004.

[32] X. Yang and G. de Veciana. Performance of peer-to-peer networks: Service capacity and role of resource sharing policies. *Performance Evaluation, Special Issue on Performance Modeling and Evaluation of P2P Computing Systems*, 2005.