Noname manuscript No.
(will be inserted by the editor)

Broadcasting a File in a Communication Network

Kai-Simon Goetzmann · Tobias Harks · Max Klimm

the date of receipt and acceptance should be inserted later

Abstract We study the problem of distributing a file, initially located at a server, among a set of n computing nodes. The file is divided into $m \geq 1$ equally-sized packets. Nodes who downloaded a packet can upload it to other nodes, possibly to many nodes in parallel. Each node, however, may receive each packet from a single source node only. The upload and download rates between nodes are constrained by node and server specific upload and download capacities. The objective is to minimize the makespan. This problem has been proposed and analyzed first by Mundinger et al. [J. Scheduling, 2008] under the assumption that uploads obey the fair-sharing principle, that is, concurrent upload rates from a common source are equal at any point in time. Under this assumption, the authors devised an optimal polynomial time algorithm for the case, where the upload capacity of the server and the nodes' upload and download capacities are all equal.

In this work, we drop the fair sharing assumption and derive an exact polynomial time algorithm for the case when upload and download capacities per node and among nodes are equal. We further show that the problem becomes strongly NP-hard for equal upload and download capacities per node that may differ among nodes, even for a single packet. For this case we devise a polynomial time $(1 + 2\sqrt{2})$-approximation algorithm. Finally, we devise two polynomial time algorithms with approximation guarantees of 9 and $2 + \lceil \log_2 \lceil n/m \rceil \rceil / m$, respectively, for the general case of m packets.

Keywords Broadcasting problem, Content distribution, File sharing, Load balancing, Makespan, Peer-to-peer, Approximation algorithm

1 Introduction

The theory and practice of computing has seen two major paradigm shifts over the past decades. First, with the rise of cloud computing services, the computations for the solution of a difficult problem are often distributed among several data centers scattered all over the world. Second, on a more microscopic level, most processor architectures have multiple cores that (ideally) solve computational problems in parallel. In both realms it is important to efficiently distribute data available at one entity (computer, processor, etc.) to all other entities of the system, or said differently, to efficiently *broadcast* data, see e.g. Armbrust et al. [3]. A widely accepted measure of efficiency is the time needed to complete the broadcast (called the makespan), leading to the minimum time broadcasting problem. The makespan objective is for instance

The results of Section 3 of this paper appeared as an extended abstract in Proceedings of the 22nd International Symposium on Algorithms and Computation [9].

Kai-Simon Goetzmann · Max Klimm
Institut für Mathematik, Technische Universität Berlin, Germany.
E-mail: {goetzmann,klimm}@math.tu-berlin.de
The research of Kai-Simon Goetzmann was supported by the Deutsche Forschungsgemeinschaft within the research training group 'Methods for Discrete Structures' (GRK 1408). The research of Max Klimm was carried out in the framework of MATHEON supported by Einstein Foundation Berlin.

Tobias Harks
School of Business and Economics, Maastricht University, Maastricht, The Netherlands.
E-mail: t.harks@maastrichtuniversity.nl

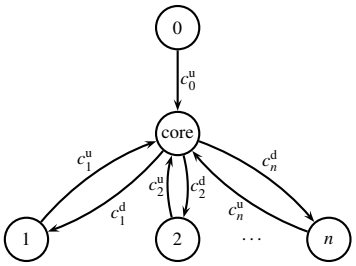


Fig. 1 Graphical representation of the file distribution problem we consider

important for cloud computing systems, where it is crucial that security updates are disseminated to every computing node as fast as possible.

In the minimum time broadcasting problem, we are given a graph, whose nodes correspond to the entities of the system. At time zero, a file is located at a designated node (the server), and the task is to disseminate the file to all nodes as fast as possible. We assume that the file is divided into m equally sized packets. At any point in time, each node that possesses a certain packet may send it with an arbitrary rate to other nodes that have not yet received this packet. As it is common in cloud computing, the nodes are connected to each other via a core network which is usually overprovisioned. Thus, the sending rates to and from other nodes (via the core) are limited by two capacities. First, for each node i there is an upload capacity c_i^u, and the sum of the sending rates of i may not exceed c_i^u. Second, every node has a download capacity c_i^d, and the sum of all incoming sending rates (involving different packets) may not exceed c_i^d.

This problem contains elements of several classical combinatorial optimization problems such as *network design*, *scheduling* and *routing*. A feasible solution (for a single packet) can be thought of as a *directed tree* rooted at the source node with the understanding that a node receives the packet from its unique parent and sends it to its children nodes. The additional feature besides finding an optimal distribution tree is to define a schedule determining the points in time when nodes actually send the packet. In contrast to classical scheduling models, however, once nodes received the packet they may send the packet to other nodes, thus, in scheduling terminology, machines can be *activated*.

Alternatively, the problem can be interpreted as a *dynamic multicommodity flow* problem in a star-shaped topology, see Figure 1. The central vertex of the network represents the core network. All nodes are connected to this vertex via a pair of antiparallel arcs, equipped with the upload and download capacities. Packets of the file can be sent from one node to another via the unique directed simple path between them. The total flow on an arc must not exceed its capacity at any point in time.

1.1 Previous Work

There is an enormous body of work on the (minimum time) broadcasting problem, multicast problem, and gossiping problem, see Hedetniemi et al. [10] for a survey. In the broadcasting problem, the task is to disseminate a file from a source node to the rest of the nodes in a given communication network as fast as possible, see Ravi [20] for an approximation algorithm. When the file needs to be disseminated only to a subset of the nodes, this task is referred to as multicasting, see Bar-Noy et al. [4] for approximation algorithms. In the gossiping problem, several nodes possess different files and the goal is to transmit every file to every node.

The usual underlying communication model for these problems is known as the telephone model: a node may send a file to at most one other node at a time, and it takes one round (i.e., one unit of time) to transfer a file. For complete graphs on $n+1$ nodes, this process terminates in $\lceil \log_2(n+1) \rceil$ rounds. It is known that for arbitrary communication graphs, the problem of computing an optimal broadcast in the telephone model is NP-hard (cf. Garey and Johnson [8]), even for 3-regular planar graphs (cf. Middendorf [16]).

Khuller et al. [11] studied the problem of broadcasting in heterogeneous networks (cf. Bar-Noy et al. [4] for the corresponding multicast problem). They consider complete graphs, but extend the telephone model by allowing the transmission time of the file to depend on the sender. They prove that it is NP-hard to minimize the makespan and present an approximation scheme.

All the above models, however, do not take into account two important features of many real-world implementations of broadcasting systems, such as cloud computing or peer-to-peer networks. First, each node typically sends the file to *multiple* nodes of the network in parallel. Second, the file is usually divided into *packets* (as the smallest indivisible unit) and a node may receive different packets from different nodes. These extensions result in structural changes of the model. In contrast to the telephone model (including the extension of Khuller et al. [11]), in our model the transfer time between any two nodes is no longer part of the instance but determined by the sending rates of a feasible solution. In fact, the model of Khuller et al. [11] for broadcasting in heterogeneous networks reduces to a special case of our model. If $c_i^{\mathrm{d}} \geq \max_{j \in N} c_j^{\mathrm{u}}$ for all $i \in N$, there always is an optimal solution in which no node will send the file to more than one other node at a time, thus, the time to transfer the file from one node to another only depends on the sender's upload capacity. In general, however, it may be beneficial to serve multiple nodes simultaneously as illustrated in the following example of a file distribution problem in a network with three nodes. Node 0 with capacity $c_0^{\mathrm{u}} = 2$ initially owns the file consisting of one packet. There are two further nodes with capacities $c_i^{\mathrm{u}} = c_i^{\mathrm{d}} = 1$, $i = 1, 2$. In the optimal solution, both nodes receive the file in parallel at a rate of 1, yielding a makespan of $M^* = 1$. Restricting nodes to upload to at most one other node at a time results in a makespan of 2.

Mundinger et al. [18] studied a peer-to-peer file distribution problem, where a file is subdivided in multiple parts and the goal is to disseminate the complete file to every peer as fast as possible. The crucial difference to our model is that they assume that the upload capacity of a node is equally shared among concurrent uploads. Under this fair-sharing assumption they prove that for homogeneous capacities (i.e. $c_i^{\mathrm{u}} = c_i^{\mathrm{d}} = 1$ for all nodes i) a simple greedy algorithm is optimal. (A similar result was also earlier reported by Kwon and Chwa [14] and Bar-Noy et al. [5], assuming the telephone model.) We prove this result *without* the fair-sharing assumption. Our proofs are quite involved, since dropping the fair sharing assumption considerably complicates matters. For heterogeneous capacities, to the best of our knowledge, neither approximation algorithms nor hardness results were known before.

When the number of parts of the file tends to infinity, one obtains the so-called fluit flow model, for which a relatively easy closed form expression of the minimum completion time can be derived, see, e.g., Ezovski et al. [6], Kumar and Ross [13], and Mehyar et al. [15]. Qiu and Srikant [19] studied a related model with stochastic arrival of peers. Fan et al. [7] assumed a finite number of parts, but assume that the number of parts is large enough such that a peer always has enough "new" parts to share with other peers.

The file distribution problem we consider has elements of a scheduling problem in which a job becomes a machine after its completion. A related problem exhibiting this property is the *freeze-tag-problem* studied in the area of robotics. Here, a set of asleep robots is placed on a graph. There is one designated robot which is awake and that can travel to other robots in order to awake them. Once a robot is awake, it may travel to other robots to awake them. The task is to awake all robots as fast as possible. Arkin et al. [2] showed the hardness of the problem in unweighted graphs and gave a constant factor approximation algorithm for the case that there is at most one robot at each node. For the general case, they obtained a $\Omega(\sqrt{\log n})$-approximation, where n is the number of robots. Koenemann et al. [12] devised a $O(\sqrt{\log n})$-approximation for the weighted case. Arkin et al. [1] studied the problem for different graph topologies. Closest to our setting, they showed that the problem is NP-complete, even for star networks. For this case, they devised a 14-approximation.

1.2 Summary of the Results and Used Techniques

Our results for a single packet. In Section 3, we study the basic situation in which a single packet is to be broadcasted. For the case of homogeneous symmetric (unit) capacities, that is, $c_i^{\mathrm{u}} = c_i^{\mathrm{d}} = 1$ for all nodes i, we show that a greedy algorithm computes an optimal solution with makespan $\lceil \log_2(n+1) \rceil$. Although similar results for the same algorithm have been obtained before, e.g. by Mundinger et al. [18], or in the broadcasting literature (Kwon and Chwa [14], and Bar-Noy et al. [5]), our result holds for a more general model as we drop the fair sharing assumption used by Mundinger et al. in the telephone model. For the case of unit node capacities and an arbitrary integer server capacity, we propose a polynomial time algorithm (that possibly splits up server capacity) and prove its optimality. We also give a closed-form expression of the minimal makespan.

If node capacities are heterogeneous and symmetric (i.e. $c_i^{\mathrm{u}} = c_i^{\mathrm{d}}$ for all nodes i) we show that the problem becomes strongly NP-hard. A key ingredient of the reduction (from 3-Partition) is the *Capacity*

Expansion Lemma (Lemma 5) that provides an upper bound on the total amount of data downloaded at any point in time.

In light of the hardness, we then study approximation algorithms. We first devise a polynomial time $2\sqrt{2}$-approximation algorithm for instances with heterogeneous, symmetric capacities in which the upload capacity of the server is larger than the download capacity of any other node. For smaller server capacities a slight modification of our algorithm gives a $(1+2\sqrt{2})$-approximation. Our algorithm which we term Scale-Fit proceeds in two phases; in a first phase, we use a time-varying resource augmentation to construct a so-called $\sqrt{2}$*-augmented solution*, that violates the capacity constraints of the nodes at any point in time by a factor of at most $\sqrt{2}$. By exploiting again our Capacity Expansion Lemma, we prove that the makespan of the augmented solution thus constructed is at most a factor of 2 away from the optimal makespan of the original instance. We then rescale the relaxed solution to obtain a feasible solution with makespan less than a factor $2\sqrt{2}$ away from the optimal makespan.

Our results for multiple packets. In Section 4, we proceed by analyzing the more challenging problem of distributing a file that is divided into $m > 1$ packets to a set of nodes with heterogeneous symmetric capacities. First, we devise an algorithm, which we call Spread-Exchange, that works in two phases. In the first phase, we use a slight variation of our Scale-Fit-Algorithm for single-packet distribution in order to send one packet to each node. At the end of phase one, each node possesses one packet of the file, but the packets owned by different nodes may differ. In fact, we strive to maximize the variety of different packets available at the nodes. In the second phase, the nodes exchange the packets among each other, in each round increasing the number of packets available at each node by one. By carefully keeping track of the proportions of the different packets available at the nodes during both phases of the algorithm, we prove that for instances for which the server has the largest capacity, Spread-Exchange achieves an 8-approximation of the optimal makespan. For smaller server capacities, a slight variation of the algorithm gives a 9-approximation.

Motivated by the fact that in many real-world instances the number of packets is large compared to the number of nodes, we also propose a different algorithm, termed Spread-Mirror-Cycle, that performs better than Spread-Exchange on these instances. The main idea is to invest more time in the first phase, in order to achieve a balanced proportion of the different packets available at the different nodes. The effort undertaken in the first phase then pays off in the second phase, for which a faster exchange procedure can be implemented. For a large number of packets the second phase dominates the makespan of the solutions of both algorithms and, thus, Spread-Mirror-Cycle performs better than Spread-Exchange. We prove that Spread-Mirror-Cycle yields a $(2+2\lceil\log_2\lceil n/m\rceil\rceil/m)$-approximation, establishing that Spread-Mirror-Cycle has a better worst-case guarantee for instances in which $n \lessapprox 2^{(7/2)m} + m$.

2 Preliminaries

An instance of the minimum time broadcasting problem is described by a tuple $I = (N, m, \mathbf{c}^{\mathrm{d}}, \mathbf{c}^{\mathrm{u}})$, where $N = \{0, \dots, n\}$ is the set of nodes, $m \in \mathbb{N}_{\geq 1}$ is the number of packets, $\mathbf{c}^{\mathrm{d}} = (c_0^{\mathrm{d}}, \dots, c_n^{\mathrm{d}}) \in \mathbb{Q}_{\geq 0}^{n+1}$ is the vector of download capacities from the core network and $\mathbf{c}^{\mathrm{u}} = (c_0^{\mathrm{u}}, \dots, c_n^{\mathrm{u}}) \in \mathbb{Q}_{\geq 0}^{n+1}$ is the vector of upload capacities to the core network. Let $[m] = \{1, 2, \dots, m\}$ be the set of packets. We will identify the server with node 0 and assume that only the server initially owns the file. See Figure 1 for an illustration. The file is of unit size; the size of each packet is thus $1/m$. A feasible solution $S = (s_{i,j}^{(k)})_{i,j\in N, k\in[m]}$ is a family of integrable functions $s_{i,j}^{(k)} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, $i, j \in N$, $k \in [m]$, where $s_{i,j}^{(k)}(t)$ denotes the rate at which node i sends packet k to node j at time t. We require that each node $i \in N$ receives each packet k from a unique node, denoted by $p_k(i) \neq i$: $s_{\ell,i}^{(k)}(t) = 0$ for all $\ell \neq p_k(i)$ and all $t \in \mathbb{R}_{\geq 0}$. In addition, only nodes that possess a packet k can send that packet with a positive rate: $s_{i,j}^{(k)}(t) = 0$ for all $i, j \in N \setminus \{0\}$, $k \in [m]$, $t \in \mathbb{R}_{\geq 0}$ with $\int_0^t \sum_{\ell\in N} s_{\ell,i}^{(k)}(\tau)\,\mathrm{d}\tau < 1/m$. Finally, the sending rates have to obey download and upload capacity constraints: $\sum_{k\in[m]} \sum_{j\in N} s_{i,j}^{(k)}(t) \leq c_i^{\mathrm{u}}$ for all $i \in N$, $t \in \mathbb{R}_{\geq 0}$, and $\sum_{k\in[m]} s_{p_k(j),j}^{(k)}(t) \leq c_j^{\mathrm{d}}$ for all $j \in N$, $t \in \mathbb{R}_{\geq 0}$. We denote by $x_i(t) = \int_0^t \sum_{k\in[m]} s_{p_k(i),i}^{(k)}(\tau)\,\mathrm{d}\tau$ the proportion of the whole file owned by node $i \in N \setminus \{0\}$ at time t. For notational convenience, we set $x_0(t) = 1$ for all $t \in \mathbb{R}_{\geq 0}$. We let $C_i = \inf\{t \in \mathbb{R}_{\geq 0} : x_i(t) = 1\}$ denote the *completion time* of node i. The makespan of a solution S is then defined as $M = \max_{i\in N\setminus\{0\}} C_i$. If an instance satisfies $c_i^{\mathrm{u}} = c_i^{\mathrm{d}} = c_j^{\mathrm{u}} = c_j^{\mathrm{d}}$ for all $i, j \in N\setminus\{0\}$ we speak of an instance with *homogeneous symmetric capacities*. If an instance satisfies $c_i^{\mathrm{u}} = c_i^{\mathrm{d}}$ for all $i \in N$ (but possibly $c_i^{\mathrm{u}} \neq c_j^{\mathrm{u}}$ for some $i, j \in N\setminus\{0\}$) we speak of *heterogeneous symmetric*

capacities. In both cases, we only write $I = (N, m, \mathbf{c})$ meaning that $\mathbf{c} = \mathbf{c}^{\mathrm{u}} = \mathbf{c}^{\mathrm{d}}$. If $m = 1$, we only write $I = (N, \mathbf{c})$.

As we will show, the minimum time broadcasting problem is in general NP-hard. A common approach in theoretical computer science to deal with such problems is to consider *approximation algorithms*. For a minimization problem $\mathcal{P}$ and $\rho > 1$, a ρ-approximation algorithm is an algorithm that, for any instance $I \in \mathcal{P}$, computes a feasible solution S in time that is polynomial in the encoding length $|I|$ of the input, such that the objective value $c(S)$ is not greater than the optimal value $\textsc{Opt}(I)$ by more than a factor of ρ, i.e. $c(S) \leq \rho \, \textsc{Opt}(I)$.

There are some particularities of the minimum time broadcasting problem regarding the polynomial running time of algorithms. In general, a solution consists of arbitrarily complicated sending rate functions $s_{i,j}^{(k)}$. Without loss of generality, we can restrict ourselves to piecewise constant functions with discontinuities only at points in time where some node starts or finishes the download of a packet. Any solution that does not have this property can be modified to do so by flattening the sending rates in the intervals where the sending and receiving nodes for each packet are constant. Specifically, let $[t_1, t_2)$ be such an interval. It is easy to verify that the new sending rates $\bar{s}_{i,j}^{(k)}$ defined as $\bar{s}_{i,j}^{(k)} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} s_{i,j}^{(k)}(\tau) \, \mathrm{d}\tau$ for all $i, j \in N$ obey the capacity constraints of the upload and download links; see also [17, Theorem 1] for a formal proof. A simple shifting argument further shows that it is without loss of generality to assume that a node starts receiving a packet only at time zero or when another node has finished receiving a packet. There are at most $m \cdot n$ of these breakpoints.

However, if the number of packets m is part of the input, encoded in binary, a polynomial time algorithm would usually only be allowed time that is polynomial in $\log m$. Given that even a simplified (optimal) solution might need space linear in m, in the context of broadcasting a divisible file we call an algorithm polynomial if it is polynomial in the encoding length of the input and m. Another reason for assuming an input size proportional to m is that in real systems each packet contains a unique piece of information and, thus, the input size is in practice proportional to m.

3 Broadcasting a Single Packet

We begin by studying the base case $m = 1$, that is, the file consists of a single packet that is to be broadcasted. Parts of the results obtained in this chapter will be reused to solve the more challenging general case $m > 1$. To increase the readability, throughout this section we drop the index k for the packet number from the notation.

This section is organized as follows. First, we propose efficient and optimal solutions for the case of *homogeneous symmetric capacities*, that is, upload and download capacities among the nodes are equal. Then, we show that for the more general case of *heterogeneous symmetric capacities* the computation of an optimal solution becomes strongly NP-hard. Finally, we give an efficient $O(n \log n)$-algorithm that approximates the optimal makespan by a factor of $2\sqrt{2}$.

3.1 Homogeneous Symmetric Capacities

In this section we consider the homogeneous symmetric setting, i.e. $c_i^{\mathrm{u}} = c_i^{\mathrm{d}} = c_i = 1$ for all $i \in N \setminus \{0\}$. The server has a capacity of $c_0^{\mathrm{u}} = c_0$.

If the server has unit upload capacity $c_0 = 1$ as well, the following greedy procedure yields an optimal solution: At each point in time, any node that already owns the file uploads it to exactly one other node, which takes one unit of time. Thus in each step, the number of nodes owning the file is doubled, resulting in a makespan of $\lceil \log_2(n+1) \rceil$. For a formal description of the procedure, see Algorithm 1.

Lemma 1 *If $c_i = 1$ for all $i \in N$, Algorithm 1 computes an optimal solution in time $O(n)$. The optimal makespan is $\lceil \log_2(n+1) \rceil$.*

Proof Throughout the algorithm, the number of nodes that own the file (including node 0) at time $t \in \mathbb{N}$ is $\min\{2^t, n\}$ (proof by induction). All completion times are integral, hence the makespan of the constructed solution is

$$M = \min\{t \in \mathbb{N} : 2^t \geq n+1\} = \min\{t \in \mathbb{N} : t \geq \log_2(n+1)\} = \lceil \log_2(n+1) \rceil \ .$$

Algorithm 1: GREEDY for identical capacities

Input: instance $I = (N, \mathbf{c})$ with $\mathbf{c} = \mathbf{1}$
Output: makespan-minimal solution S

$s_{i,j}(\tau) := 0 \ \forall \tau \geq 0$;
$A := \{0\}$, $P := N \setminus \{0\}$, $t := 0$;
while $P \neq \emptyset$ **do**
  $A' := \emptyset$;
  forall the $i \in A$ **do**
    Choose $j \in P$;
    $s_{i,j}(t) := 1$ for all $t \in [t, t+1)$;
    $A' := A' \cup \{j\}$, $P := P \setminus \{j\}$;
  end
  $A := A \cup A'$, $t := t + 1$;
end

To prove that this is best possible, consider an optimal solution $S^* = (s^*_{i,j})_{i,j \in N}$ with the corresponding completion times $C^*_1, \ldots, C^*_n$. We will modify this solution such that in the time interval $[0, 1)$ only one node is served, completing the transfer at time 1, without increasing the makespan. Iterating this argument yields a solution with the same structure as the one constructed by Algorithm 1, establishing the claim. Let the nodes be indexed such that $C^*_1 = \min_{i=1,\ldots,n} C^*_i$. We set the rate vector s' of the modified solution as follows:

$$s'_{0,1}(t) = \begin{cases} 1, & \text{for } t \in [0,1) \\ 0, & \text{for } t \geq 1 \end{cases}$$

$$s'_{0,i}(t) = \begin{cases} 0, & \text{for } t \in [0,1) \\ \frac{x^*_i(C^*_1)}{C^*_1 - 1}, & \text{for } t \in [1, C^*_1) \\ s^*_{0,i}(t), & \text{for } t \geq C^*_1 \end{cases} \qquad \forall\, i \in N \setminus \{1\} : p(i) = 0$$

$$s'_{i,j}(t) = s^*_{i,j}(t), \quad \text{for all other } i, j, t$$

Without loss of generality, we can assume $s^*_{i,j}(t) = \frac{1}{C^*_1} \int_0^{C^*_1} s^*_{i,j}(\tau)\, \mathrm{d}\tau$ for all $t \in [0, C^*_1)$ and all $i, j \in N$; see [17, Theorem 1]. It then holds that $x^*_i(C^*_1) = s^*_{0,i}(0) \cdot C^*_1$ for $i \in N \setminus \{1\}$ with $p(i) = 0$. Using this, we can show that in the modified solution the upload capacity of node 0 is obeyed: For $t \in [1, C^*_1)$,

$$\begin{aligned} \sum_{i \in N \setminus \{1\} : p(i) = 0} s'_{0,i}(t) &= \frac{1}{C^*_1 - 1} \sum_{i \in N \setminus \{1\} : p(i) = 0} x^*_i(C^*_1) \\ &= \frac{C^*_1}{C^*_1 - 1} \sum_{i \in N \setminus \{1\} : p(i) = 0} s^*_{0,i}(0) \\ &= \frac{1}{1 - \frac{1}{C^*_1}} \Bigg(\underbrace{\sum_{i \in N} s^*_{0,i}(0)}_{\leq 1} - s^*_{0,1}(0) \Bigg) \\ &\leq \frac{1}{1 - s^*_{0,1}(0)} \Big(1 - s^*_{0,1}(0) \Big) \\ &= 1 \,. \end{aligned}$$

The inequality above also yields $s'_{0,i}(t) \leq 1$ for all $i \in N \setminus \{1\}$ with $p(i) = 0$, so download capacities are obeyed as well since $p(i) = 0$ implies $s^*_{j,i}(t) = 0$ for all $j \neq 0$, $t \in \mathbb{R}_{\geq 0}$. Finally, at time C^*_1 every node $i \in N$ owns the same fraction of the file as in the original solution, namely $x^*_i(C^*_1)$. Since in S^* no node completes its download before time C^*_1, and in the modified solution after time C^*_1 the original rates are used, no completion of a download is delayed, and hence the makespan is not increased. □

We now consider the case where server capacity c_0 is an arbitrary integer, and start with a proof that in this case there always is an optimal solution that uses *fair-sharing*, i.e. whenever node 0 serves several nodes simultaneously, all of them are served with equal rate. The proof is quite involved, and the integrality condition on c_0 is crucial. To see this, consider an example with $c_0 = 3/2$ and $n = 4$. In an optimal solution, the server serves node 1 in $[0, 1)$ and node 2 in $[1, 2)$ with a rate of 1 each and node 3 with a rate of $1/2$ in

$[0, 2)$. Node 4 is served by node 1 in $[1, 2)$, resulting in a makespan of $M^* = 2$. The best solution that uses fair-sharing, however, has a makespan of $M = 7/3$: The server serves nodes 1 and 2 in $[0, 4/3)$ with a rate of $3/4$ each, and node 3 and 4 are both served at a rate of 1 in $[4/3, 7/3)$ by the server and node 1.

Lemma 2 *For any instance $I = (N, \mathbf{c})$ with $c_0 \in \mathbb{N}$ and $c_i = 1 \; \forall \; i = 1, \ldots, n$, there always exists an optimal solution that uses* fair sharing.

Proof For an arbitrary instance with the demanded properties, consider an optimal solution S^* with the corresponding completion times $C_1^*, \ldots, C_n^*$. W.l.o.g. assume that the nodes are indexed such that $\{1, 2, \ldots, k\} = \{i \in N : p(i) = 0\}$ and $C_1^* = \ldots = C_{k'}^* < C_{k'+1}^* \leq \ldots \leq C_k^*$. Let $N_1 = \{1, \ldots, k'\}$. For $i = 1, \ldots, k$, let m_i be the makespan for serving all nodes that are directly or indirectly served by node i, i.e.

$$m_i = \max\{C_j^* - C_i^* : j \in N, p^\ell(j) = i \text{ for some } \ell \in \mathbb{N}\},$$

where $p^\ell(j)$ denotes the ℓ-fold application of the function p on j. By Lemma 1 we can assume that these nodes are served according to Algorithm 1 (otherwise we modify the solution to get this structure without increasing the makespan). In particular we get that $m_i \in \mathbb{N}$ for all $i = 1, \ldots, k$. The makespan of the given optimal solution is $M^* = \max_{i=1,\ldots,k} C_i^* + m_i$. Finally let $M_1 = \max_{i \in N_1} m_i$.

We will show that we can either first serve all nodes in N_1 with equal rates and delay the serving of the other nodes until N_1 is completed, or we can increase the size of N_1 by adding further nodes to it, and then serve the nodes in this increased set jointly with equal rates. One of these two modifications can be done without increasing the makespan. This argument can be iterated until node 0 never serves several nodes simultaneously using different rates.

We prove this by a case distinction. To describe the modified solution we use primed variables s', C', x' and so on. Whenever $s'_{i,j}(t)$ is not specified for some i, j, t, it is equal to the original value $s^*_{i,j}(t)$.

Case 1: $k \leq c_0$.
In this case, we let node 0 serve the nodes $1, \ldots, k$ at a rate of 1 in the time interval $[0, 1)$. Obviously no node completes its download later than in the given solution, and fair-sharing is used all the time.

In the following cases we always assume that $k > c_0$.

Case 2: $M_1 \leq m_{i_0}$ for some $i_0 \in \{k' + 1, \ldots, k\}$.
We add one node to N_1, serving nodes $1, \ldots, k' + 1$ with equal rates in the time interval $[0, C^*_{k'+1})$:

$$\forall \; i \in 1, \ldots, k' + 1 : \qquad s'_{0,i}(t) = \begin{cases} \sum_{j=1}^{k'+1} s^*_{0,j}(t) \,/\, (k' + 1) & \forall \; t \in [0, C^*_{k'+1}) \,, \\ 0 & \text{otherwise} \,. \end{cases}$$

Then nodes $1, \ldots, k' + 1$ all complete their download at time $C^*_{k'+1}$:

$$x'_i(C^*_{k'+1}) = \int_0^{C^*_{k'+1}} s'_{0,i}(\tau) \; \mathrm{d}\tau = \frac{1}{k' + 1} \sum_{j=1}^{k'+1} \underbrace{\int_0^{C^*_{k'+1}} s^*_{0,j}(\tau) \; \mathrm{d}\tau}_{=1} = 1 \quad \forall i = 1, \ldots, k' + 1 \; .$$

The download capacities of all nodes are satisfied since at any point t in time $s'_{0,i}(t) = \frac{1}{k'+1} \sum_{j=1}^{k'+1} s^*_{0,j} \leq \max_{j=1,\ldots,k'+1} s^*_{0,j} \leq 1$. The same is true for the upload capacity of the server, since it uploads with the same overall rate as in the original solution at each point in time.

Furthermore, this modification does not increase the makespan: Since $M_1 \leq m_{i_0}$, for all $i \in N_1$ we have $C'_i + m_i = C^*_{k'+1} + m_i \leq C^*_{i_0} + m_{i_0} \leq M^*$, so delaying the completion of nodes in N_1 does not influence the makespan.

We now turn to the cases where $M_1 > m_i$ for all $i = k' + 1, \ldots, k$. Note that by Lemma 1 we know that $m_i \in \mathbb{N}$ for all i, hence we can assume $M_1 \geq m_i + 1$ for all $i > k'$ in the subsequent cases.

Case 3: $k' \geq c_0$ and $M_1 \geq m_i + 1$ for all $i > k'$.
In this case, we modify the solution such that node 0 first serves all nodes in N_1 at full capacity, delaying the serving of all other nodes until those in N_1 have completed their downloads. More formally, let $t_1 = k' \,/\, c_0$, and set

$$\forall \; i \in N_1 : \qquad s'_{0,i}(t) = \begin{cases} c_0 \,/\, k' & \forall \; t \in [0, t_1) \\ 0 & \text{otherwise.} \end{cases}$$

$$\forall \; i = k' + 1, \ldots, k : \qquad s'_{0,i}(t) = 0 \quad \forall \; t \in [0, t_1) \; .$$

It is obvious that in $[0, t_1)$ all capacities are obeyed and $C'_i = t_1 \leq C^*_i$ for all $i \in N_1$.

To specify how the remaining nodes are served, let $t_2 = \max\{C^*_1, t_1 + 1\}$. We serve nodes $k' + 1, \ldots, k$ in such a way that the fraction of the file they own at time t_2 is the same as in the original solution, i.e. $x'_i(t_2) = x^*_i(t_2)$:

$$\forall\ i = k' + 1, \ldots, k: \qquad s'_{0,i}(t) = \frac{x^*_i(t_2)}{t_2 - t_1} \quad \forall\ t \in [t_1, t_2)\ .$$

Since $x^*_i(t_2) \leq 1$ and $t_2 - t_1 \geq 1$, this obeys the download capacities of all nodes. Also the upload capacity of the server is obeyed, since in the time interval $[0, t_2)$ it sends the same volume as in the original solution:

$$\int_0^{t_2} \sum_{i \in N} s'_{0,i}(\tau)\, \mathrm{d}\tau \;=\; \sum_{i=1}^{k'} \int_0^{t_1} \frac{c_0}{k'}\, \mathrm{d}\tau + \sum_{i=k'+1}^{k} \int_{t_1}^{t_2} \frac{x^*_i(t_2)}{t_2 - t_1}\, \mathrm{d}\tau \;=\; \sum_{i=1}^{k'} \underbrace{t_1 \cdot \frac{c_0}{k'}}_{=1=x^*_i(t_2)} + \sum_{i=k'+1}^{k} x^*_i(t_2) \;=\; \sum_{i=1}^{k} x^*_i(t_2)\ ,$$

and the overall rate the server sends at is c_0 at the beginning and constant afterwards, hence it is at most c_0 at all times.

Nodes in N_1 and nodes with $C^*_i \geq t_2$ complete their download no later than in the original solution. If there is a node $i \in \{k' + 1, \ldots, k\}$ with $C^*_i < t_2$, then $t_2 > C^*_1$ and thus $t_2 = t_1 + 1$, and this node i completes its download exactly one time unit after the nodes in N_1. Since $M_1 \geq m_i + 1$ we get $C'_i + m_i = t_1 + 1 + m_i \leq C^*_1 + M_1 \leq M^*$, therefore the makespan is not increased.

Case 4: $k' < c_0$ and $M_1 \geq m_i + 1$ for all $i > k'$.
In this case, we add nodes $k' + 1, \ldots, c_0$ to N_1 and serve these nodes jointly at full capacity in the time interval $[0, 1)$:

$$\forall\ i = 1, \ldots, c_0: \qquad s'_{0,i}(t) = \begin{cases} 1 & \forall\ t \in [0, 1) \\ 0 & \text{otherwise.} \end{cases}$$
$$\forall\ i = c_0 + 1, \ldots, k: \qquad s'_{0,i}(t) = 0 \quad \forall\ t \in [0, 1)\ .$$

For the serving of the remaining nodes we set $t_2 = \max\{C^*_{c_0}, 2\}$ and serve nodes $c_0 + 1, \ldots, k$ such that $x'_i(t_2) = x^*_i(t_2)$ by setting

$$\forall\ i = c_0 + 1, \ldots, k: \qquad s'_{0,i}(t) = \frac{x^*_i(t_2)}{t_2 - 1} \quad \forall\ t \in [1, t_2)\ .$$

Feasibility follows similarly to Case 3, as well as the fact that the makespan is not increased. □

We continue by proving that the server always serves groups of c_0 nodes jointly, possibly except for some start interval $[0, t_1)$, $t_1 \in \mathbb{R}_{\geq 0}$ where a group of at least c_0 and most $2c_0$ nodes is served.

Lemma 3 *For any instance $I = (N, \mathbf{c})$ with $c_0 \in \mathbb{N}$ and $c_i = 1\ \forall\ i = 1, \ldots, n$, there always exists an optimal solution that uses fair sharing where the server never serves more than c_0 nodes simultaneously except for some interval $[0, t_1]$, $t_1 \in \mathbb{R}_{\geq 0}$ in which at least c_0 and at most $2c_0$ nodes are server simultaneously until they are completed.*

Proof For an arbitrary instance with the demanded properties, consider an optimal solution S^*. Using Lemma 2, we can assume w.l.o.g. that the set $\{i \in N : p(i) = 0\}$ of nodes served by the server is partitioned into sets $N_1, \ldots, N_k$ such that all nodes $i \in N_j$ are served simultaneously by the server at equal rates in the time interval $[t_{j-1}, t_j]$, where $t_0 = 0$ and $t_j = \sum_{\ell=1}^{j} \max\{1, |N_\ell| \,/\, c_0\}$ for $j = 1, \ldots, k$. We can further assume that $|N_j| \geq c_0$ for $j = 1, \ldots, k-1$ (otherwise we can add nodes from N_{j+1} to N_j without increasing the makespan) and $|N_j| < 2c_0$ for $j = 1, \ldots, k$ (otherwise we split N_j into two sets and serve c_0 nodes in $[t_{j-1}, t_{j-1} + 1]$ and the remaining $|N_j| - c_0$ nodes in $[t_{j-1} + 1, t_j]$).

For $j = 1, \ldots, k$, we denote by $\tilde{N}_j$ the set of nodes that are directly or indirectly served by a node in N_j, and by M_j the makespan needed for this, i.e.,

$$\begin{aligned} \tilde{N}_j &= \{i \in N : p^\ell(i) = i' \text{ for some } i' \in N_j, \ell \in \mathbb{N}\}\ , \\ M_j &= \max\nolimits_{i \in \tilde{N}_j} C^*_i - t_j\ , \end{aligned} \tag{1}$$

Algorithm 2: Extended Greedy for integral server capacity

Input: instance $I = (N, \mathbf{c})$ with $c_0 \in \mathbb{N}, c_i = 1$ for all $i \in N \setminus \{0\}$
Output: makespan-minimal solution S

$h := \left\lceil \log_2\left(\frac{n}{c_0} + 1\right)\right\rceil$; // Choose $h \in \mathbb{N}$ such that $n \in [c_0(2^h - 1), c_0(2^{h+1} - 1))$
if $n \geq c_0(2^h - 1 + 2^{h-1})$ **then** // n in the second half of $[c_0(2^h - 1), c_0(2^{h+1} - 1))$
  Choose $N_1 \subseteq N$ with $|N_1| = c_0$;
  $k := h + 1$;
else if $n < c_0(2^h - 1 + 2^{h-1})$ **then** // n in the first half of $[c_0(2^h - 1), c_0(2^{h+1} - 1))$
  Choose $N_1 \subseteq N$ with $|N_1| = \left\lceil \frac{n - c_0(2^{h-1}-1)}{2^{h-1}} \right\rceil$;
  $k := h$;
end
$t_1 := |N_1| / c_0$;
In $[0, t_1)$, node 0 serves nodes in N_1;
for $t = 0, \ldots, k-2$ **do**
  Choose $N_{t+2} \subseteq N$ of nodes not yet served, with $|N_{t+2}| = c_0$;
  In $[t_1 + t, t_1 + t + 1)$, node 0 serves nodes in N_{t+2}, any completed node $i \in N \setminus \{0\}$ serves one other node that is not yet served (if there still is one);
end

where again $p^\ell(j)$ denotes the ℓ-fold application of the function p on j. As in the proof of Lemma 2, we can assume that $M_j \in \mathbb{N}$ for all $j = 1, \ldots, k$. We now prove by induction that the solution can be modified such that $|N_j| \leq c_0$ for $j = 2, \ldots, k$, without increasing the makespan. For the base case, assume $|N_2| > c_0$. We decrease N_2 to contain only c_0 nodes.

If $M_1 \leq M_2 + 1$, the $|N_2| - c_0$ removed nodes are served by the server jointly with the nodes in N_1. This delays the completion of nodes in N_1 to time $\frac{|N_1|+|N_2|-c_0}{c_0} = t_2 - 1$, but using $M_1 \leq M_2 + 1$ it does not increase the makespan since the new makespan of any node in $\tilde{N}_1$ can be bounded by $t_2 - 1 + M_1 \leq t_2 + M_2$. The case $M_1 \geq M_2 + 2$ is more involved. Here, the $|N_2| - C_0$ removed nodes will be served by nodes in $\tilde{N}_2$. For this, the makespan M_2 has to be increased by at most one. To see this note that $|\tilde{N}_2| \leq |N_2| \cdot 2^{M_2}$. With $|N_2'| = c_0$, in $M_2 + 1$ time units we can serve up to $c_0 \cdot 2^{M_2+1} = 2c_0 \cdot 2^{M_2} > |N_2| \cdot 2^{M_2}$ nodes through nodes in N_2', so we can serve all nodes in $\tilde{N}_2$ (including the nodes from N_2). The overall makespan is not increased by this, since the serving of N_2' is now completed at time $t_2' = t_1 + 1$, and hence all nodes in $\tilde{N}_2$ are served at time $t_2' + M_2 + 1 = t_1 + M_2 + 2 \leq t_1 + M_1 \leq M^*$.

For the inductive step, assume $|N_j| > c_0$ for some $j = 3, \ldots, k$. By the induction hypothesis, we know that $|N_\ell| = c_0$ for $\ell = 2, \ldots, j-1$ and thus $t_{j-1} = t_\ell + j - \ell - 1$ for all $\ell = 1, \ldots, j-1$. With this and a distinction between the cases $M_\ell \leq M_j + j - \ell$ and $M_\ell \geq M_j + j - \ell + 1$, the inductive step follows similarly to the base case. □

Summing up, we have shown so far that there always is an optimal solution where the server at the beginning serves a set of nodes N_1, $c_0 \leq |N_1| < 2c_0$, in $|N_1| / c_0$ time units, then some sets $N_2, \ldots, N_{k-1}$ with $|N_j| = c_0 \ \forall j \geq 2$, in one time unit each, and finally a set N_k containing at most c_0 nodes, in another unit of time.

We are now ready to present an exact algorithm for this special case and prove its correctness. Intuitively speaking, the algorithm is an extended greedy procedure that attempts to balance the sub-makespans M_j of the different sets N_j of nodes served by the server. Set $h = \lceil \log_2(n/c_0 + 1) \rceil$. If $n < c_0(2^h - 1 + 2^{h-1})$, let $|N_1| = \lceil (n - c_0(2^{h-1} - 1))/2^{h-1} \rceil$ and $k = h$, otherwise let $|N_1| = c_0$ and $k = h + 1$. Let $|N_j| = c_0$ for $j = 2, \ldots, k$. The server serves N_1 in $[0, |N_1|/c_0)$ and N_j in $[|N_1|/c_0 + j - 2, |N_1|/c_0 + j - 2)$. Meanwhile, any node that finishes its download starts serving other nodes greedily. Details are listed in Algorithm 2.

Theorem 4 *For any instance $I = (N, \mathbf{c})$ with $c_0 \in \mathbb{N}$ and $c_i = 1 \ \forall \ i = 1, \ldots, n$, Algorithm 2 yields an optimal solution. The optimal makespan is*

$$M^* = \begin{cases} h - 1 + \frac{1}{c_0}\left\lceil \frac{n - c_0(2^{h-1}-1)}{2^{h-1}} \right\rceil & \text{if } n \in [c_0(2^h - 1), c_0(2^h - 1 + 2^{h-1})) \\ h + 1 & \text{if } n \in [c_0(2^h - 1 + 2^{h-1}), c_0(2^{h+1} - 1)) \end{cases},$$

where $h = \lceil \log_2(n/c_0 + 1) \rceil$.

Proof W.l.o.g. we can assume that in the solution produced by Algorithm 2, $|\tilde{N}_j| = c_0 \cdot 2^{k-j}$ for all $j = 2, \ldots, k$ (with $\tilde{N}_j$ as defined in (1)), i.e. all sets $\tilde{N}_j$ for $j \geq 2$ are as big as possible. If this is not the case

we can move nodes from $\tilde{N}_1$ to these sets. The lemmas above state that there also is an optimal solution with this structure and $c_0 \le |N_1| < 2c_0$. In the argumentation below, we therefore restrict to solutions of this structure.

The integer h is chosen such that $c_0(2^h - 1) \le n < c_0(2^{h+1} - 1)$. We first consider the case where $n \ge c_0(2^h - 1 + 2^{h-1})$. Our algorithm sets $k = h + 1$ and $|N_1| = c_0$. The maximum number of nodes that can be served like this is $c_0(2^{h+1} - 1) > n$, so indeed all nodes are served within a makespan of $h + 1$. Assume that in an optimal solution only $k = h$ sets are served by the server. The maximum number of nodes served in this way is $|N_1| \cdot 2^{h-1} + c_0(2^{h-1} - 1) < c_0(2^h + 2^{h-1} - 1) \le n$, so not all nodes can be served, contradiction! Therefore in an optimal solution it must hold that $k = h+1$ and $|N_1| \ge c_0$, proving that the solution produced by Algorithm 2 is optimal.

If $n < c_0(2^h - 1 + 2^{h-1})$, the size of N_1 in an optimal solution (with $|\tilde{N}_j| = c_0 \cdot 2^{k-j}$ for $j \ge 2$) has to be chosen such that $|\tilde{N}_1| = n - c_0(2^{h-1} - 1) \le |N_1| \cdot 2^{h-1}$, i.e.

$$|N_1| = \min\left\{\ell \in \mathbb{N} : \ell \ge \frac{n - c_0(2^{h-1} - 1)}{2^{h-1}}\right\} = \left\lceil \frac{n - c_0(2^{h-1} - 1)}{2^{h-1}} \right\rceil ,$$

which is exactly how the algorithm chooses N_1. Moreover, the algorithm sets $k = h$. It is obvious that choosing $k \le h - 1$ does not lead to a solution where all nodes are served, and choosing $k \ge h + 1$ leads to a solution with a makespan of at least $h + 1$. The solution produced by our algorithm has a makespan of $t_1 + k - 1 \le h + 1$, where we used $t_1 \le 2$ and $k = h$. This proves the correctness of Algorithm 2. □

3.2 Heterogeneous Symmetric Capacities

In this section, we consider the case of heterogeneous and symmetric capacities. First, we show that the minimum time broadcasting problem for heterogeneous symmetric capacities is strongly NP-hard, even for an indivisible file. Then, we devise an algorithm that approximates the optimal makespan by a factor $1 + 2\sqrt{2}$. Our hardness and approximation results rely on a useful lemma that bounds the work that has to be done in any feasible solution. For a feasible solution, let $u_i(t_1, t_2)$ and $z_i(t_1, t_2)$ denote the total upload and the idleness of node i in time interval $[t_1, t_2]$, defined as $u_i(t_1, t_2) = \int_{t_1}^{t_2} \sum_{j \in N} s_{i,j}(\tau)\, \mathrm{d}\tau$ and $z_i(t_1, t_2) = \int_{\max\{t_1, C_i\}}^{t_2} (c_i - \sum_{j \in N} s_{i,j}(\tau))\, \mathrm{d}\tau$. For $t \ge 0$, we define $X(t) = \sum_{i \in N} x_i(t)$ and $Z(t) = \sum_{i \in N} z_i(0, t)$. We obtain the immediate equality $X(t_2) - X(t_1) = \sum_{i \in N} u_i(t_1, t_2)$ for all $0 \le t_1 \le t_2$.

Lemma 5 (Capacity Expansion Lemma) *Let $I = (N, \mathbf{c})$ be an instance of the minimum time broadcasting problem with an indivisible file, and let $c = \max_{i \in N} c_i$. Then, for all solutions S of I, the following two statements hold:*

1. $u_i(t_1, t_2) + z_i(t_1, t_2) \le \max\left\{0, \left(t_2 - t_1 - \frac{1 - x_i(t_1)}{c_i}\right) c_i\right\} = \max\left\{0, (t_2 - t_1)\, c_i - 1 + x_i(t_1)\right\}$ *for all nodes $i \in N$ and all times $0 \le t_1 < t_2$.*
2. $X(\frac{k}{c}) + Z(\frac{k}{c}) \le 2^k$ *for all $k \in \mathbb{N}$. This inequality is strict if there is $i \in N$ with $c_i < c$ and $0 < C_i \le \frac{k}{c}$.*

Proof To see 1., note that for any node i with $C_i \le t_1$, the upload rate (integrand of the total upload) and the idle rate (integrand of the idleness) sum up to c_i for all $t \in [t_1, t_2]$, thus $u_i(t_1, t_2) + z_i(t_1, t_2) \le (t_2 - t_1)\, c_i$. If $x_i(t_1) < 1$, node i needs at least $\frac{1 - x_i(t_1)}{c_i}$ time units to finish its download, thus only a time interval of length $t_2 - t_1 - \frac{1 - x_i(t_1)}{c_i}$ remains for the upload and the claimed inequality follows.

We prove 2. by induction over k. The inequality is trivial for $k = 0$ since initially only the server holds the file. So, let us assume that for $k \in \mathbb{N}$, we have $X(\frac{k-1}{c}) + Z(\frac{k-1}{c}) \le 2^{k-1}$ and that this inequality is strict if there is $i \in N$ with $c_i < c$ and $0 < C_i \le \frac{k-1}{c}$. Using 1., we obtain

$$\begin{aligned} X(\tfrac{k}{c}) - X(\tfrac{k-1}{c}) + Z(\tfrac{k}{c}) - Z(\tfrac{k-1}{c}) &= \sum_{i \in N} u_i(\tfrac{k-1}{c}, \tfrac{k}{c}) + z_i(\tfrac{k-1}{c}, \tfrac{k}{c}) \\ &\le \sum_{i \in N} \max\left\{0, \tfrac{c_i}{c} - 1 + x_i(\tfrac{k-1}{c})\right\} \\ &\le X(\tfrac{k-1}{c}) , \end{aligned} \tag{2}$$

where we use $c_i \le c$. Rearranging terms and using the induction hypothesis, we obtain

$$X(\tfrac{k}{c}) + Z(\tfrac{k}{c}) \le 2X(\tfrac{k-1}{c}) + Z(\tfrac{k-1}{c}) \le 2^k . \tag{3}$$

To finish the proof, let us assume that there is $i \in N$ with $c_i < c$ and $0 < C_i \leq \frac{k}{c}$. If $C_i \leq \frac{k-1}{c}$, using the induction hypothesis, we obtain $X(\frac{k-1}{c}) < 2^{k-1}$ and, by (3), $X(\frac{k}{c}) < 2^k$. If, on the other hand, $C_i \in (\frac{k-1}{c}, \frac{k}{c}]$, then the inequality (2) is satisfied strictly and we obtain $X(\frac{k}{c}) < 2^k$ by (3).

3.2.1 Hardness

We are now ready to prove strong NP-hardness of the minimum time broadcasting problem.

Theorem 6 *The minimum time broadcasting problem for heterogeneous symmetric capacities is strongly NP-hard, even for* $m = 1$.

Proof We reduce from 3-Partition. An instance of 3-Partition is given by a multiset $P = \{k_1, \ldots, k_n\}$ of $n = 3\mu$, $\mu \in \mathbb{N}$, positive integers. The goal is to partition P into μ subsets $P_0, \ldots, P_{\mu-1}$ such that the sum of all integers within each subset is equal. The decision whether such partition exists remains NP-complete in the strong sense even if $B/4 < k_i < B/2$ for all $k_i \in P$, where $B = \sum_{i=1}^{n} k_i/\mu$ is integer, see Garey and Johnson [8]. We first construct a modified 3-Partition instance in which μ is a power of two and larger or equal to 16. To this end, let $\ell = \max\{\lceil \log_2 \mu \rceil, 4\}$ and set $\mu' = 2^\ell$. We define the new instance by $P' = P \cup \bigcup_{k=\mu+1}^{\mu'} \{B/2, B/2\}$. The goal is to partition P' into μ' subsets $P_0, \ldots, P_{\mu'-1}$ such that the sum of all integers within each subset is equal. Note that $|P'|$ might not be equal to $3\mu'$, but this does not hurt our following arguments. Because the integer $B/2$ can only go with another integer $B/2$ in a "yes"-instance of the modified 3-Partition problem, the modified 3-Partition instance P' admits a solution if and only if the original instance P admits a solution.

Given the modified instance P' of 3-Partition, we construct an instance I of the minimum time broadcasting problem as follows: For every $k_i \in P'$, we introduce a *master element node* i_0 with capacity $c_{i_0} = k_i$ and $2\ell k_i - 2$ *element nodes* $i_1, \ldots, i_{2\ell k_i - 2}$ with capacity $c_{i_k} = k_i \,/\, (\ell k_i - 1)$ for all $k \in \{1, \ldots, 2\ell k_i - 2\}$. For $j \in \{0, \ldots, \mu' - 1\}$ we introduce a *subset node* s_j with capacity $c_{s_j} = B$. Subset node s_0 initially owns the file. Note that 3-Partition is strongly NP-complete, i.e., is NP-complete even if the integers k_i are represented unary. For unary encoded integers of the 3-Partition instance, we obtain a polynomial reduction.

We claim that the optimal makespan of I is less or equal $\ell/B + \ell$ if and only if P' is a yes-instance. For the "if"-part, we construct a solution S where first all subset nodes are served, and then each subset node serves those master element nodes that correspond to elements in one of the sets of the partition. Afterwards all element nodes are served. More formally, in the first $1/B$ time units subset node s_0 sends the file to the first subset node s_1 with a rate of $s_{s_0,s_1} = B$. After $1/B$ time units, the first subset node has completed the download of the file. Starting at time $1/B$ subset nodes s_0 and s_1 send the file to the subset nodes s_2 and s_3, respectively, each with a rate of $s_{s_0,s_2} = s_{s_1,s_3} = B$. Continuing in this fashion, at time ℓ/B all μ' subset nodes own the file but none of the other nodes has received any data. Let $P_0^*, \ldots, P_{\mu'-1}^*$ be a solution of P'. In the remaining ℓ time units, subset node s_j, $j \in \{0, \ldots, \mu' - 1\}$, sends the file to every master element node i_0 with $k_i \in P_j^*$, at a rate of $s_{s_j,i_0} = k_i$. This is feasible since $P_0^*, \ldots, P_{\mu'-1}^*$ is a solution of P' and thus $\sum_{i \in S_j^*} k_i = B$. The download of each master element node is finished after $1/k_i$ time units at time $\ell/B + 1/k_i$. At this point in time, master element node i_0 starts to send data to $\ell k_i - 1$ of the corresponding element nodes i_j, $j \in \{1, \ldots, \ell k_i - 1\}$, at a rate of $s_{i_0,i_j} = k_i \,/\, (\ell k_i - 1)$. The remaining $k_i - 1$ element nodes i_j, $j \in \{\ell k_i, 2\ell k_i - 2\}$, are served by the subset node s_j with rate $s_{s_j,i_j} = k_i \,/\, (\ell k_i - 1)$. The download of the file by the element nodes starts at time $\ell/B + 1/k_i$ and needs additional $(\ell k_i - 1) \,/\, k_i$ time units, resulting in a total makespan of $\ell/B + \ell$.

It remains to be shown that the optimal makespan of the broadcasting instance I is strictly larger than $\ell/B + \ell$ in the case that P' is a no-instance. We first argue that no element node i_k, $k \in \{1, \ldots, 2\ell k_i - 2\}$, will upload the file to any other node. Both the upload and the download of the complete file by element node i_k requires at least $(\ell k_i - 1) \,/\, k_i = \ell - 1/k_i$ time units. We calculate

$$2\left(\ell - \frac{1}{k_i}\right) \geq \frac{4\ell}{3} + \frac{8}{3} - \frac{2}{k_i} \geq \frac{4\ell}{3} + \frac{2}{3} > \frac{4\ell}{3} \geq \ell\left(\frac{1}{B} + 1\right),$$

where we use $\ell \geq 4$. This value is larger than the assumed makespan. Hence, we can assume that only the subset nodes (including the server s_0) and the master element nodes upload the file to other nodes. In order to find a solution of I with $M \leq \frac{\ell}{B} + \ell$, it is necessary to find a partial solution S serving the subset nodes and the master element nodes with $Z(\frac{\ell}{B} + \ell) \geq \sum_{i \in P'} (2\ell k_i - 2)$, otherwise the idleness of subset nodes and master

element nodes does not suffice to serve all element nodes until time $\frac{\ell}{B} + \ell$. We proceed by showing that no such partial solution exists. Let us first assume that $x_{s_j}(\frac{\ell}{B}) = 1$ for every subset node s_j. Using Lemma 5, the inequality $X(\frac{\ell}{B}) + Z(\frac{\ell}{B}) \leq \mu'$ holds, thus $Z(\frac{\ell}{B}) = 0$ and $x_{i_0}(\frac{\ell}{B}) = 0$ for every master element node i_0. We calculate

$$\begin{aligned} Z(\tfrac{\ell}{B} + \ell) &= \sum_{i \in N} z_i(\tfrac{\ell}{B}, \tfrac{\ell}{B} + \ell) \\ &= \mu' \ell B + \sum_{k_i \in P'} k_i(\tfrac{\ell}{B} + \ell - C_{i_0}) - \underbrace{\sum_{j \in N} \sum_{k_i \in P'} \int_{\frac{\ell}{B}}^{\frac{\ell}{B}+\ell} s_{j,i_0}(\tau)\,\mathrm{d}\tau}_{=|P'|} \\ &= \mu' \ell B + \sum_{k_i \in P'} \Big(k_i(\tfrac{\ell}{B} + \ell - C_{i_0}) - 1 \Big) . \end{aligned}$$

For the completion time C_{i_0} of a master element node i_0, we obtain the inequality $C_{i_0} \geq \ell/B + 1/k_i$. We claim that there is at least one master element node i_0' for which this inequality is strict. For a contradiction, suppose that $C_{i_0} = \ell/B + 1/k_i$ for all $k_i \in P'$. This implies that every master element node i_0 downloads with its full rate $s_{p(i_0),i_0} = c_{i_0}$. Let us define $P_j = \{k_i : p(i_0) = s_j\}$. As subset node s_j has capacity $c_{s_j} = B$ and serves all master element nodes i_0 with $k_i \in S_j$ with full rate, we derive that $\sum_{k_i \in S_j} k_i \leq B$. Thus, S_j is a solution to the 3-Partition instance P', a contradiction. Consequently, there is a master element node i_0' with $C_{i_0'} > \ell/B + 1/k_i$, establishing that

$$Z(\tfrac{\ell}{B} + \ell) < \mu' \ell B + \sum_{k_i \in P'} (\ell k_i - 2) = \sum_{k_i \in P'} (2\ell k_i - 2) .$$

We are left with the case that at least one of the subset nodes has not finished the download of the file at time ℓ/B, that is, there is $j' \in \{1, \ldots, \mu' - 1\}$ with $x_{s_{j'}}(\ell/B) < 1$. We calculate

$$\begin{aligned} Z(\tfrac{\ell}{B} + \ell) &\leq Z(\tfrac{\ell}{B}) + \sum_{j=0}^{\mu'-1} B\,(\tfrac{\ell}{B} + \ell - C_{s_j}) + \sum_{k_i \in P'} k_i\,(\tfrac{\ell}{B} + \ell - C_{i_0}) - \sum_{i \in N} \sum_{j \in N} \int_{\frac{\ell}{B}}^{\frac{\ell}{B}+\ell} s_{i,j}(\tau)\,\mathrm{d}\tau \\ &\leq Z(\tfrac{\ell}{B}) + \sum_{k_i \in P'} 2\ell k_i + \sum_{j=0}^{\mu'-1} B\,(\tfrac{\ell}{B} - C_{s_j}) + \sum_{k_i \in P'} k_i\,(\tfrac{\ell}{B} - C_{i_0}) - \sum_{i \in N} \Big(1 - x_i(\tfrac{\ell}{B})\Big) \qquad (4) \\ &= \sum_{k_i \in P'} (2\ell k_i - 1) - \mu' + \sum_{j=0}^{\mu'-1} B\,(\tfrac{\ell}{B} - C_{s_j}) + \sum_{k_i \in P'} k_i\,(\tfrac{\ell}{B} - C_{i_0}) + \underbrace{X(\tfrac{\ell}{B}) + Z(\tfrac{\ell}{B})}_{\leq \mu'} \\ &\leq \sum_{k_i \in P'} (2\ell k_i - 1) + \sum_{j=0}^{\mu'-1} B\,(\tfrac{\ell}{B} - C_{s_j}) + \sum_{k_i \in P'} k_i\,(\tfrac{\ell}{B} - C_{i_0}) , \qquad (5) \end{aligned}$$

where (4) stems from the fact that each node $i \in N$ still needs to download a proportion of $1 - x_i(\ell/B)$ of the file. For each subset node s_j with $x_{s_j}(\ell/B) < 1$ we have $C_{s_j} \geq \ell/B + (1 - x_{s_j}(\ell/B))\,/\,B$, and for each master element node with $x_{i_0}(\ell/B) < 1$ we have $C_{i_0} \geq \ell/B + (1 - x_{i_0}(\ell/B))\,/\,k_i$. We distinguish two cases.

First Case: At least one of the master element nodes has finished the download at time ℓ/B. Then, inequality (5) is strict and we obtain

$$\begin{aligned} Z(\tfrac{\ell}{B} + \ell) &< \sum_{k_i \in P'} (2\ell k_i - 1) - \sum_{j=0}^{\mu'-1} \Big(1 - x_{s_j}(\tfrac{\ell}{B})\Big) - \sum_{k_i \in P'} \Big(1 - x_{i_0}(\tfrac{\ell}{B})\Big) \\ &= \sum_{k_i \in P'} (2\ell k_i - 1) + \underbrace{X(\tfrac{\ell}{B})}_{\leq \mu'} - \mu' - |P'| \\ &\leq \sum_{k_i \in P'} (2\ell k_i - 2) . \end{aligned}$$

Second Case: All master element nodes and at least one of the subset nodes still have not finished the downloads at time ℓ/B.

Then, the upload capacity of the other subset nodes is not sufficient to serve all master element nodes with their full download rate, thus, one of the inequalities $C_{i_0} \geq \frac{\ell}{B} + \frac{1-x_{i_0}(\ell/B)}{k_i}$ is satisfied strictly. As a consequence, we obtain $Z(\ell/B + \ell) < \sum_{k_i \in P'}(2\ell k_i - 2)$. □

3.2.2 Approximation

In this section, we devise an algorithm that runs in time $O(n \log n)$ and computes a solution with makespan no larger than $(1 + 2\sqrt{2})M^*$, where M^* is the optimal makespan. We first consider the case $c_0 \geq c_i$ for all $i = 1, \ldots, n$, for which we will show a $2\sqrt{2}$-approximation. Then, we will use this result to obtain a $(1 + 2\sqrt{2})$-approximation for arbitrary server capacities. The additional 1 in the approximation arises as possibly we first have to transfer the file to the node having the largest capacity. This takes at most M^* time, thus, giving an additional 1 in the approximation guarantee.

Before we present details of the algorithm, we give a high-level picture of our approach. The intrinsic difficulty in proving any approximation guarantee is to obtain lower bounds on the optimal makespan. Let us recall the inequality shown in Lemma 5: for *any* solution, node i's contribution to the upload in time interval $[t_1, t_2]$ is bounded by $u_i(t_1, t_2) \leq \max\{0, c_i\,(t_2 - t_1) - 1 + x_i(t_1)\}$. To exploit this capacity bound, our algorithm should fulfill two properties: it should send the file to nodes with large capacity as soon as possible, and it should avoid idle time as long as possible. To achieve this, we use the concept of *time-varying resource augmentation*. For $\lambda \geq 1$, we call a possibly infeasible solution $\tilde{S}$ a *λ-augmented* solution, if at any point in time the original capacities are not exceeded by more than a factor of λ, i.e. $\sum_{j \in N} s_{i,j}(t) \leq \lambda c_i^{\mathrm{u}}$ and $s_{p(i),i}(t) \leq \lambda c_i^{\mathrm{d}}$ for all $i \in N$, $t \in \mathbb{R}_{\geq 0}$. We will show that there is an efficiently computable *$\sqrt{2}$-augmented* solution that satisfies the above mentioned properties. This allows us to apply the Capacity Expansion Lemma, and we get that the makespan $\tilde{M}$ of $\tilde{S}$ is not larger than $2M^*$. Rescaling the augmented solution $\tilde{S}$ to obtain a feasible solution S, we get an additional factor of $\sqrt{2}$.

Now, we explain our algorithm (termed Scale-Fit, see Algorithm 3 for a formal definition) in more detail. Let the nodes be labeled such that $c_0 \geq \cdots \geq c_n$. We choose an index k such that

$$\frac{c_0}{\sqrt{2}} \leq \sum_{j=1}^{k} c_j \leq \sqrt{2} c_0 \tag{6}$$

is satisfied (line 12 of Algorithm 3), and assign nodes $1, \ldots, k$ to node 0. Note that there always exists such an index k unless node 0 can serve all nodes simultaneously at full download rate. This is formally proven in the following lemma, which we give for the slightly weaker assumption $c_0 \geq c_1/\sqrt{2}$.

Lemma 7 *Let $c_1 \geq \cdots \geq c_n$ and $c_0 \geq \frac{c_1}{\sqrt{2}}$. If $\sum_{j=1}^{n} c_j > c_0$, then there is $k \leq n$ with $\frac{c_0}{\sqrt{2}} \leq \sum_{j=1}^{k} c_j \leq \sqrt{2} c_0$.*

Proof If $\frac{c_0}{\sqrt{2}} \leq c_1$, there is nothing left to show. Otherwise, let

$$k' = \max\left\{\ell \in \{1, \ldots, n\} : \textstyle\sum_{j=1}^{\ell} c_j < \frac{c_0}{\sqrt{2}}\right\}.$$

Since we assume $c_1 < c_0/\sqrt{2}$ and $\sum_{j=1}^{n} c_j > c_0$, we have $1 \leq k' \leq n - 1$. We set $k = k' + 1$. By definition, $\sum_{j=1}^{k} c_j \geq c_0/\sqrt{2}$. In addition, we have $\sum_{j=1}^{k} c_j \leq 2 \sum_{j=1}^{k'} c_j < \sqrt{2} c_0$. □

Given the choice of k according to (6), we now explain the scaling of the capacities. We distinguish two cases. If the total capacity of the downloading nodes $1, \ldots, k$ exceeds the capacity of the uploading node 0, we augment the upload capacity to match the total download capacity. Otherwise the downloaders' capacities are scaled by a common factor in order to fit the uploader's capacity. Formally, the capacities of nodes $1, \ldots, k$ are increased by a factor of $\alpha = \max\{1, c_0/\sum_{j=1}^{k} c_j\}$ and that of node 0 by a factor of $\beta = \sum_{j=1}^{k} \alpha c_j / c_0$ (lines 13 and 15). Note that by (6), the scaling factor is not greater than $\sqrt{2}$ in either case.

At time 0, all k nodes start downloading from node 0 with their full (possibly augmented) capacity (line 20). Whenever a node finishes its download, the algorithm rescales the augmented capacity (either that of the downloader or that of the uploader) to the original level (line 21). After rescaling, we proceed serving new nodes by the respective unused capacities of the parent and the finished node according to (6), breaking ties arbitrarily. The process stops as soon as the total capacity of the remaining nodes is less than the currently available upload capacity. From this point on, all remaining nodes are served simultaneously at full download rate. For the formal details of the Scale-Fit procedure, see Algorithm 3.

The choice of the augmentation factors and the rescaling procedure ensure the invariant that the augmented capacities do not exceed the original capacities by more than a factor of $\sqrt{2}$.

Algorithm 3: Scale-Fit

Input: instance $I = (N, \mathbf{c})$ with $c_0 = \max_{i \in N} c_i$
Output: $2\sqrt{2}$-approximate solution S of I

Sort nodes non-increasingly with respect to their capacities, i.e. $c_1 \geq c_2 \geq \cdots \geq c_n$;
$Q := \{(0, 0, c_0)\}$; // server has unused upload capacity c_0 at time 0
$k := 1$; // node with largest capacity not served yet
$b := 1$; // largest augmentation factor
while $k \leq n$ **do**
  $(t, i, c) := \arg\min_{(t', i', c') \in Q} t'$;
  $Q := Q \setminus \{(t, i, c)\}$;
  if $\sum_{j=k}^{n} c_j \leq \sqrt{2}c$ **then** // i serves all remaining nodes
    $k' := n$;
    $\alpha := 1$; // augmentation factor of downloader
  else
    Choose k' such that $c/\sqrt{2} \leq \sum_{j=k}^{k'} c_j \leq \sqrt{2}c$;
    $\alpha := \max\{1, c / \sum_{j=k}^{k'} c_j\}$; // augmentation factor of downloader
  end
  $\beta := (\sum_{j=k}^{k'} \alpha c_j) / c$; // augmentation factor of uploader
  $b := \max\{b, \alpha, \beta\}$; // update largest augmentation factor
  $\tilde{A}_j := t \; \forall j \in \{k, \ldots, k'\}$; // time when j starts downloading
  $\tilde{C}_j := t + 1/(\alpha c_j) \; \forall j \in \{k, \ldots, k'\}$; // time when j finishes downloading
  $p(j) := i \; \forall j \in \{k, \ldots, k'\}$; // parent of nodes $k, \ldots, k'$ is j
  $\tilde{s}_{i,j}(t) := \alpha c_j \; \forall j \in \{k, \ldots, k'\} \; \forall t \in [\tilde{A}_j, \tilde{C}_j)$; // sending rate from i to j
  $Q := Q \cup \bigcup_{j=k}^{k'} \{(\tilde{C}_j, j, c_j) \cup (\tilde{C}_j, i, \alpha c_j / \beta)\}$; // rescale capacity
  $k := k' + 1$;
end
forall the $i \in N \setminus \{0\}$ **do**
  $A_i := \tilde{A}_i b, \; C_i := \tilde{C}_i b, \; s_{p(i),i}(t) := \tilde{s}_{p(i),i}(\tilde{A}_i)/b \; \forall t \in [A_i, C_i)$; // Rescale $\tilde{S}$
end

Lemma 8 Scale-Fit *computes a* $\sqrt{2}$*-augmented solution* $\tilde{S}$.

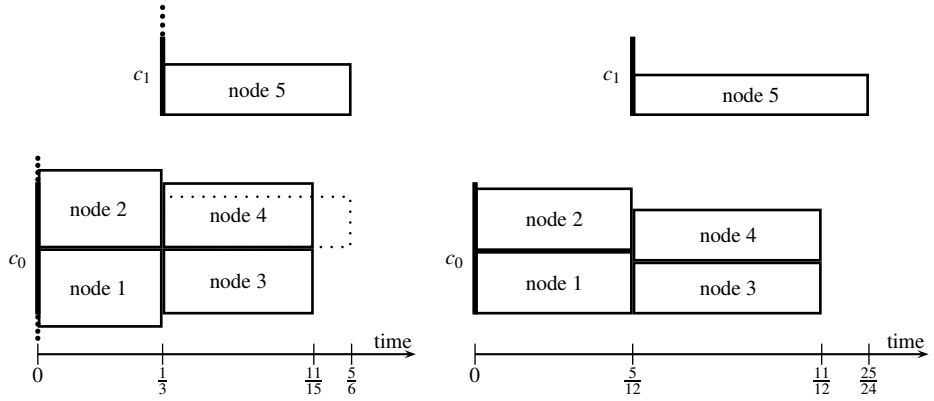
Proof Obviously, all download constraints are violated by a factor of at most $\sqrt{2}$. To see that this holds also for all upload constraints, let us say that every node i initially *owns* its upload capacity c_i and passes it in fractions $\alpha c_j/\beta$ to the nodes $j \in \{k_1, \ldots, k_2\}$ downloading from i at time C_i. Note that $\sum_{j=k_1}^{k_2} \alpha c_j/\beta = c_i$.

Now consider a node ℓ that owns a fraction c of node i's upload capacity and finishes downloading from node i. The fraction c of node i's upload capacity is now passed in subfractions $\alpha c_j/\beta$ from node ℓ to the nodes $j \in \{k_3, \ldots, k_4\}$ that start the download at that time from node i. As $\sum_{j=k_3}^{k_4} \alpha c_j/\beta = c$, no upload capacity is generated. Also, no node violates the upload capacity it owns by a factor more than $\sqrt{2}$, and the claimed result follows. □

Before we formally analyse the performance of Scale-Fit, let us illustrate the algorithm by an example:

Example 9 Consider an instance with 6 nodes and capacities $c_0 = 5$, $c_1 = 3$, $c_2 = 3$, $c_3 = 5/2$, $c_4 = 2$, and $c_5 = 2$. We first describe how to construct the augmented solution $\tilde{S}$. At time 0, the upload capacity of node 0 is augmented by a factor of $6/5$ in order to serve nodes 1 and 2 at their full download capacities. These downloads are completed at time $1/3$. At that time, the rescaled upload capacities of the parent of nodes 1 and 2 (i.e., node 0) can be used separately to serve further nodes. The $5/2$ units of capacity (previously assigned to node 1) are now assigned to node 3, which downloads with full capacity without any augmentation. The remaining $5/2$ units of capacity are assigned to node 4, whose download capacity is augmented by a factor of $5/4$. At the same time, node 1 starts serving node 5 without any augmentation because node 5 is the only remaining node. The highest augmentation factor is $5/4$. Thus, we rescale all rates by $4/5$ and obtain the feasible solution S. See Fig. 2 for an illustration.

We now turn to the approximation guarantee of our algorithm. For this, let $\tilde{S}$ be the $\sqrt{2}$-augmented solution generated by Scale-Fit, and let T_0 denote the first point in time in this solution when a node does not fully use its upload capacity, that is, $T_0 = \min\{t \geq 0 : \exists\, i \in N \text{ with } \tilde{C}_i \leq t \text{ and } \sum_{j \in N} \tilde{s}_{i,j}(t) < c_i\}$. We first show that T_0 is a lower bound on the optimal makespan.



(a) Augmented solution $\tilde{S}$ (b) Feasible solution S

Fig. 2 (a) Augmented solution $\tilde{S}$ and (b) feasible solution S of Example 9.

Lemma 10 *Let $I = (N, \mathbf{c})$ be an instance of the minimum time broadcasting problem with $c_0 \geq c_i \ \forall i = 1, \ldots, n$, and let $\tilde{S}$ be the $\sqrt{2}$-augmented solution generated by* Scale-Fit. *Then $T_0 \leq M^*$, where M^* is the optimal makespan.*

Proof Let us first consider the case $T_0 < \tilde{M}$. For a contradiction, suppose $M^* < T_0$ and fix an optimal solution S^*. We have $\tilde{X}(T_0) < X^*(T_0) = n$. Let k_0 be the node with smallest index that has not finished the download at time T_0, that is, $k_0 = \min\{i \in N : \tilde{C}_i > T_0\}$. Such a node exists, since we assume $T_0 < \tilde{M}$. If $k_0 = 1$, then we obtain the inequality $T_0 < 1/c_1$, which is a lower bound on M^*, and there is nothing left to show. So we assume $k_0 > 1$ and consider the point in time $T_1 = T_0 - 1/c_{k_0}$. For a contradiction, let us assume that $\tilde{X}(T_1) \geq X^*(T_1)$.

For the $\sqrt{2}$-augmented solution $\tilde{S}$ returned by Scale-Fit, we have $\tilde{x}_{k_0}(T_1) = 0$, and since the nodes start downloading in the order of their indices, we also get $\tilde{x}_i(T_1) = 0$ for all $i \geq k_0$. By construction, every node $i \in N$ with $\tilde{x}_i(T_1) > 0$ receives data with download rate $d_i = \alpha c_i \geq c_i$ until time $\tilde{C}_i$. Using $\tilde{z}_i(T_1, T_0) = 0$, we obtain

$$\tilde{u}_i(T_1, T_0) \geq \left(\frac{1}{c_{k_0}} - \frac{1 - \tilde{x}_i(T_1)}{d_i}\right) c_i = \frac{c_i}{c_{k_0}} + \frac{c_i}{d_i}\Big(\tilde{x}(T_1) - 1\Big) \geq \frac{c_i}{c_{k_0}} + \tilde{x}_i(T_1) - 1$$

for all $i \leq k_0$ with $\tilde{x}_i(T_1) > 0$, and $\tilde{u}_i(T_1, T_0) \geq 0$ for all other nodes. Referring to Lemma 5 (1.), we calculate

$$\begin{aligned} X^*(T_0) - X^*(T_1) &\leq \sum\nolimits_{i \in N} \max\left\{0, \frac{c_i}{c_{k_0}} - 1 + x_i^*(T_1)\right\} \\ &= \sum\nolimits_{i \in N : x_i^*(T_1) > 1 - c_i/c_{k_0}} \left(\frac{c_i}{c_{k_0}} - 1 + x_i^*(T_1)\right) \\ &\leq \sum\nolimits_{i < k_0 : x^*(T_1) > 1 - c_i/c_{k_0}} \left(\frac{c_i}{c_{k_0}} - 1\right) + X^*(T_1)\ , \end{aligned}$$

where we use $c_i/c_{k_0} \leq 1 \ \forall i \geq k_0$. We get $X^*(T_0) - X^*(T_1) \leq \sum_{i<k_0}(c_i \,/\, c_{k_0} - 1) + \tilde{X}(T_1) \leq \tilde{X}(T_0) - \tilde{X}(T_1)$, a contradiction. We conclude $X^*(T_1) > \tilde{X}(T_1)$. Applying the same line of argumentation for T_1 instead of T_0, we derive the existence of $T_2 = T_1 - 1/c_{k_1}$ for some $k_1 \in N$ with $X^*(T_2) > \tilde{X}(T_1)$. We can iterate this argument until we reach T_ℓ, with the property that $X^*(T_\ell) > \tilde{X}(T_\ell)$ and $k_\ell = \min\{i \in N : \tilde{C}_i > T_\ell\} = 1$. This is a contradiction since in the time interval $[0, T_\ell]$ only the server can upload and its upload rate in $\tilde{S}$ is not smaller than that in S^*. We conclude that our initial assumption that $T_0 > M^*$ was wrong, finishing the proof for the case $T_0 < \tilde{M}$. If $T_0 = \tilde{M}$, we can use the same line of argumentation for $T_0 - \epsilon$ instead of T_0, where $\epsilon > 0$ is arbitrary. Thus we obtain $T_0 \leq M^*$ also for that case. □

We are now ready to prove the approximation guarantee of Scale-Fit.

Theorem 11 *For the minimum time broadcasting problem with heterogeneous symmetric capacities and an indivisible file, the following holds:*

1. *If $c_0 \geq c_1$,* Scale-Fit *is a $2\sqrt{2}$-approximation.*
2. *If $c_0 < c_1$, uploading the file to the node 1 and applying* Scale-Fit *is a $(1 + 2\sqrt{2})$-approximation.*

The running time of Scale-Fit *is $O(n \log n)$.*

Proof We first show 1. By construction, node n determines the makespan and starts its download not after T_0. Hence, the makespan of the $\sqrt{2}$-augmented solution $\tilde{S}$ is not larger than $T_0 + 1/c_n \leq 2M^*$, where we use Lemma 10 and the fact that $1/c_n$ is a lower bound on the optimal makespan. For rational input, the largest factor, with which a capacity constraint is violated, is strictly smaller than $\sqrt{2}$. Dividing all sending rates by that factor, we obtain a feasible solution with makespan smaller than $2\sqrt{2}M^*$.

To see 2., note that the server needs $1/c_0 \leq M^*$ time units to transfer the file to node 1. We then treat the instance as an instance where node 1 is the server, i.e. we do not let node 0 upload the file to any other node except node 1. Using the same arguments as in the proof of Lemma 10, we derive that this takes at most $2\sqrt{2}M^*$ additional time units, implying the claimed approximation factor.

The complexity of the algorithm is as follows: In a first step, the nodes are sorted with respect to their capacities (line 1 of Algorithm 3), which takes time $O(n \log n)$. The most expensive operations within the loop (lines 5–23) are adding elements to Q (line 21) and retrieving the minimal element from Q (line 6). In total, there are $O(n)$ of these operations, and if Q is a priority queue, all operations can be executed in time $O(\log n)$. Therefore the overall running time of the algorithm is $O(n \log n)$. □

4 Broadcasting Multiple Packets

We now turn to the more general case that the file is divided into $m \in \mathbb{N}_{\geq 1}$ packets. The file is still of unit size, therefore the download of one packet at a rate of c takes $\frac{1}{cm}$ time units. We present two efficient algorithms. The first one uses the Scale-Fit algorithm, developed in the last section, as a subroutine and yields a 9-approximation. The second algorithm has an approximation guarantee of $2 + 2\lceil \log_2 \lceil n/m \rceil \rceil / m$, which is not constant, but better than 9 if the number of packets is large compared to the number of nodes.

4.1 A Constant Factor Approximation

In this section, we present a constant factor approximation algorithm for broadcasting a file that is divided into multiple packets. The algorithm, which we term Spread-Exchange, proceeds in two phases. In the first phase, Spread, the Scale-Fit algorithm is used to provide each node with a packet. The packets are distributed in a regular manner that allows an efficient Exchange of the packets among the nodes. After this second phase, each node has received the complete file.

To describe the algorithm in detail, we need the notion of the *distribution tree T* corresponding to a solution S constructed by Scale-Fit. The node set of T consists of all nodes $i \in N$ in the communication network, plus a *dummy* node $d(i)$ for each node $i \in N \setminus \{0\}$. To avoid confusion, we will call the nodes $i \in N$ *proper* nodes in this context. The dummy node $d(i)$ corresponds to the capacity of $p(i)$ that is released when node i finishes its download. The root of the tree is the server node. The children of a proper node i are all proper nodes j that are served by node i immediately after it completed its download (for the server this is at the beginning), plus their dummy nodes $d(j)$. The children of a dummy node $d(i)$ are those proper nodes j that are served by $p(i)$ using the capacity of $p(i)$ that is released when i finishes its download, plus the dummy nodes $d(j)$ of those nodes. With each node i or $d(i)$ we associate its capacity c_i and $c_{d(i)}$, respectively. We also write $C_{d(i)} = C_i$.

We now describe the algorithm step by step, justifying each step at the same time. As a preprocessing, we round up all capacities to powers of two. A solution for the modified instance yields a 2-augmented solution, which we can scale back to get a feasible solution without losing more than a factor of two in the approximation guarantee. If all capacities are powers of two, no scaling within the algorithm Scale-Fit is needed, in particular the released upload capacity of an uploader is always the same as the newly created upload capacity of the downloader, i.e. $c_i = c_{d(i)}$ in our distribution tree.

We run Scale-Fit on the modified instance, such that afterwards each proper node owns exactly one packet. To specify which packet is send in each download, let S denote the computed solution and T the

corresponding distribution tree. For $v \in V(T)$ define level(v) to be the length of the path from v to 0 in T. The server sends packet k to all proper nodes i with $p(i) = 0$ and level(i) $= k$, and packet m to all proper nodes i with $p(i) = 0$ and level(i) $> m$. Any node $i \in N \setminus \{0\}$ sends the only packet it owns. Assume for now that the height of the tree is sufficiently big for all packets to be sent, i.e., $h(T) := \max_{v \in V(T)} \text{level}(v) \geq m$.

For the Exchange part of the algorithm, we first extend the distribution tree by *virtual* nodes in order to obtain a complete tree. We will discuss later how to model this with the real nodes. The following observation shows that by adding at most one layer of virtual nodes, we can make all leaves of the tree have the same level:

Observation 12 For any two nodes $v, w \in V(T)$ (possibly dummy nodes) with level(v) $<$ level(w), it holds that $c_v \geq c_w$ and $C_v < C_w$. In particular, if there is a node at level $k + 2$, then all nodes at level k use their full capacity. (Proof by induction over level(v).)

For every leaf v with level(v) $= h(T) - 1$, we append a virtual node $\tilde{v}$ with capacity $c_{\tilde{v}} = c_v$, and add this node to N. We also assume that $\tilde{v}$ owns the same packet as v. The extended tree is complete (except for some missing dummy nodes, but this is not relevant in the second phase of the algorithm), i.e. all leaves have the same level. Moreover, the packets are distributed in a very regular fashion, as the following lemma states:

Lemma 13 *Let $\bar{c} = \sum_{i \in N} c_i$, and for $k \in [m]$ let $N_k = \{i \in N \setminus \{0\} : i \text{ owns packet } k\}$ and $\bar{c}_k = \sum_{i \in N_k} c_i$. Then*

$$\bar{c}_k = \begin{cases} \bar{c}/2^k & \text{for } k = 1, \ldots, m-1 \\ \bar{c}/2^{m-1} - c_0 & \text{for } k = m\,. \end{cases}$$

Proof Let R be the set of the children of the server in T that are proper nodes, i.e. $R = \{i \in N : \text{level}(i) = 1\}$. For any proper or dummy node r, denote by $T(r)$ the subtree of T that has r as a root.

The nodes that own packet 1 are exactly the nodes in the subtrees that have a node from R as a root, thus $\bar{c}_1 = \sum_{r \in R} \sum_{i \in N \cap T(r)} c_i$. Since T is complete, in any subtree the capacities of the proper nodes double with every level. This implies that for any proper node i, the capacities of proper nodes contained in $T(i)$ and the capacities of proper nodes contained in $T(d(i))$ differ by c_i only. For any $r \in R$ we thus obtain

$$\sum_{i \in N \cap T(r)} c_i = c_r + \sum_{i \in N \cap T(d(r))} c_i\,.$$

Therefore,

$$\begin{aligned} \bar{c} &= c_0 + \sum_{r \in R} \left(\sum_{i \in N \cap T(r)} c_i + \sum_{i \in N \cap T(d(r))} c_i \right) \\ &= \sum_{r \in R} c_r + \sum_{r \in R} \left(\left(2 \sum_{i \in N \cap T(r)} c_i \right) - c_r \right) \\ &= 2\bar{c}_1\,. \end{aligned}$$

This shows the claim for $k = 1$. We can now recursively apply this argument in the subtrees below nodes in $d(R)$: Let $1 < k < m$, and let R denote the proper nodes of level k that are served by the server. Then $\bar{c}_k = \sum_{r \in R} \sum_{i \in N \cap T(r)} c_i$. By induction,

$$\begin{aligned} \bar{c} &= \bar{c}_1 + \ldots + \bar{c}_{k-1} + c_0 + \sum_{r \in R} \left(\sum_{i \in N \cap T(r)} c_i + \sum_{i \in N \cap T(d(r))} c_i \right) \\ &= \frac{\bar{c}}{2} + \ldots + \frac{\bar{c}}{2^{k-1}} + \sum_{r \in R} c_r + \sum_{r \in R} \left(\left(2 \cdot \sum_{i \in N \cap T(r)} c_i \right) - c_r \right) \\ &= (1 - 2^{-(k-1)})\bar{c} + 2\bar{c}_k\,, \end{aligned}$$

hence $\bar{c}_k = \bar{c}/2^k$. For $k = m$, we have $\bar{c}_m = \sum_{r \in R} \left(\sum_{i \in N \cap T(r)} c_i + \sum_{i \in N \cap T(d(r))} c_i \right)$, and thus

$$\bar{c} = (1 - 2^{-(m-1)})\bar{c} + c_0 + \bar{c}_m\,,$$

or equivalently $\bar{c}_m = \bar{c}/2^{m-1} - c_0$, as claimed. □

For the Exchange part of the algorithm, we treat the server as a node that owns only packet m. We carry out $m-1$ rounds of exchange. In each round, the total capacity of nodes that own a specific packet is doubled. Thus after round k, all nodes own packet k, and this packet can be ignored in all further rounds. After round $m-1$, all nodes own all packets, and the file is completely distributed.

In round k, we assign the nodes that own packet k to nodes that own other packets $k' > k$. We do this with a straightforward greedy algorithm, see Algorithm 5 for details. As long as the largest capacity c_i of a node i that owns packet k is larger than the capacity of at least one of the other nodes, we match this node i to a subset $\tilde{N}$ of the other nodes, where $\tilde{N}$ is chosen in such a way that the capacities in this set sum up to c_i. This is always possible since all capacities are powers of two. As soon as the maximum capacity of an unmatched node that owns packet k is smaller than the minimum capacity of the unmatched nodes without packet k, we proceed the other way round – a single node i that does *not* own packet k is matched to a set of nodes that do own packet k, where the capacities in this set sum up to c_i. Using Lemma 13, it is straightforward to show that all nodes will get matched this way.

Every node now sends the packet it owns at a rate of $c_{\min}/2$ to its matched node(s). If the receiving node already gets this packet from a different node we simply omit this transfer. The way we matched the nodes guarantees that each node uses at most half of its capacities for this.

Finally, we describe how to model the virtual nodes. As a consequence of Observation 12, all virtual nodes are children of *real* (proper or dummy) nodes. We associate each virtual node with a real proper node, namely with the first proper node on the path from the virtual node to the root. By this, for each proper node i, the capacities of its associated virtual nodes sum up to at most c_i. We use the free capacities of the real proper nodes to do the work of the virtual nodes: Each real proper node sends and receives at a rate of $c_{\min}/2$ what its associated virtual nodes would have had to send and receive.

A detailed description of the whole procedure Spread-Exchange can be found in Algorithm 4. We now prove that it is a 9-approximation algorithm.

Theorem 14 *If $m \leq h(T)$,* Spread-Exchange *yields a 9-approximation in time $O(n(\log n + m))$.*

Proof We first prove the approximation guarantee. In a first step, if $c_0 < c_{\max}$, we send the whole file to a node i with $c_i = c_{\max}$. This takes $1/c_0 \leq \textsc{Opt}$ time units. We then treat this situation as a new instance, where the former server is excluded and i is the new server.

Next, we observe that any feasible solution decomposes into m solutions with a single packet. The makespan of all these solutions is bounded from below by the makespan $\textsc{Opt}_1$ of an optimal solution for one packet, and Scale-Fit yields a solution with a makespan of at most $2\textsc{Opt}_1$.

Finally, in the Exchange part of our algorithm, we use $m-1$ rounds of length $\frac{2}{mc_{\min}}$, to distribute all packets. Since a node with minimal capacity has to receive the whole file, we know that $\textsc{Opt} \geq 1/c_{\min}$.

In the end, since we only computed a 2-augmented solution, we scale all rates by $1/2$, doubling the makespan of our solution. We can thus conclude that the makespan M of the computed solution is bounded by

$$M \leq \frac{1}{c_0} + 2\left(2\textsc{Opt}_1 + \frac{2(m-1)}{mc_{\min}}\right) \leq 9\textsc{Opt}\,.$$

The running time of the algorithm is determined by the two phases. The Spread part of the algorithm executes the Scale-Fit procedure and thus has a running time of $O(n \log n)$. The appending of the virtual nodes can be done by iterating over the set N once, and thus only takes linear time. It also increases the set of nodes by at most n.

For the Exchange part of the algorithm, the matching in each round can be computed in linear time, simply by traversing a sorted list of the nodes twice, once for the case where a single node in N_k is matched with several nodes in N_{-k}, and once for the converse case. This is due to the fact that all capacities are powers of two. Since there are $m-1$ rounds, the total complexity of computing the matchings is $O(nm)$. Finally, in each round, each node sends to at most two other nodes, thus also computation of the sending rates takes a total time of $O(nm)$. □

Note that if the server has the largest capacity among all nodes, the approximation factor of Spread-Exchange reduces to 8. Moreover, in the situation where in the beginning the packets are spread all over the network, we still have a 9-approximation: In the beginning all packets are send to a node with largest capacity, which then acts as a server. This sending does not take longer than Opt.

Algorithm 4: SPREAD-EXCHANGE for heterogeneous capacities and multiple packets

Input: instance $I = (N, \mathbf{c}, m)$ with $c_1 \geq c_2 \geq \ldots \geq c_n$
Output: 9-approximate solution S

$t_0 := 0$;
round up all capacities to powers of 2 ;
if $c_0 < c_1$ **then**
  $s^{(k)}_{0,1}(t) = c_0/k$ for all $t \in [0, 1/c_0), k \in [m]$; // move server
  remove node 0, rename node i to node $i-1$ for all $i \geq 1$;
  $t_0 := 1/c_0$;
end
run SCALE-FIT from time t_0 on // spread
  node 0 sends packet k to all nodes of level $k \leq m$, and packet m to all nodes of level $k > m$;
  let T be the resulting distribution tree and t_1 the latest completion time ;
end
forall the *leaves v of T with* $\text{level}(v) = h(T) - 1$ **do**
  append virtual node $\tilde{v}$ to v ;
  give $\tilde{v}$ the same packet that is owned by v ;
  $N := N \cup \{\tilde{v}\}$;
  $a(\tilde{v}) :=$ first $i \in N$ on the path from v to 0 in T ;
end
for $k = 1, \ldots, m-1$ **do** // exchange
  $N_k := \{i \in N \setminus \{0\} : i \text{ owns packet } k\}$, $N_{-k} := N \setminus N_k$;
  Greedy assignment for N_k and N_{-k};
  forall the $i \in N_k$ **do**
    forall the *j assigned to i* **do**
      if *i is virtual* **then** $i := a(i)$;
      if *j is virtual* **then** $j := a(j)$;
      $s^{(k)}_{i,j}(t) := c_{\min}/2 \quad \forall\, t \in [t_1, t_1 + \frac{2}{mc_{\min}})$, unless j already receives k from a different node ;
      let $k' > k$ be a packet that is owned by j ;
      $s^{(k')}_{j,i}(t) := c_{\min}/2 \quad \forall\, t \in [t_1, t_1 + \frac{2}{mc_{\min}})$, unless i already receives k' from a different node ;
    end
  end
  $t_1 := t_1 + \frac{2}{mc_{\min}}$;
end
scale down all sending rates by a factor of 2 ;

Algorithm 5: Greedy assignment

Input: sets $A = \{a_1, \ldots, a_r\}$, $B = \{b_1, \ldots, b_s\}$ of powers of two with $\sum_{a \in A} a = \sum_{b \in B} b$
Output: assignment $(a_i, B_i)_{i \in I}$, $(A_j, b_j)_{j \in J}$ s.t. $A = (\dot\bigcup_{i \in I} a_i) \dot\cup (\dot\bigcup_{j \in J} A_j)$, $B = (\dot\bigcup_{i \in I} B_i) \dot\cup (\dot\bigcup_{j \in J} b_j)$, $a_i = \sum_{b \in B_i} b \;\forall i$, $b_j = \sum_{a \in A_j} a \;\forall j$

$M = \emptyset$;
while $A \neq \emptyset$ **do**
  $s = \arg\max_{t \in A \cup B} t$;
  if $s \in A$ **then**
    choose $B' \subseteq B$ with $\sum_{b \in B'} b = s$;
    $M := M \cup (s, B')$, $A := A \setminus \{s\}$, $B := B \setminus B'$;
  else
    assign s to a set $A' \subseteq A$ with $\sum_{a \in A'} a = s$;
    $M := M \cup (A', s)$, $B := B \setminus \{s\}$, $A := A \setminus A'$;
  end
end
return M;

It remains to be shown that also in the case where $m > h(T)$, we get a constant approximation factor. In this case we simply run our algorithm several times, until all packets are distributed. The approximation guarantee remains the same:

Theorem 15 SPREAD-EXCHANGE *yields a* 9*-approximation. Its worst-case complexity is* $O(nm \log n)$.

Proof To prove the approximation factor, let $k < m$ be the number of packets distributed in the first run, and let $p, r \in \mathbb{N}$ be such that $m = pk + r$, with $r < k$. We then know that in the SPREAD part of the algorithm, the maximum number of subsequent uploads by the server is k, each taking at most $\frac{1}{mc_{\min}}$ time units, and

no download starts strictly after the server has finished its last upload. Thus the makespan of our Scale-Fit solution is at most $\frac{k}{mc_{\min}} + \frac{1}{mc_{\min}} = \frac{k+1}{mc_{\min}}$. The Exchange part of our algorithm is executed p times for k packets ($\frac{2(k-1)}{mc_{\min}}$ time units each) and once for r packets ($\frac{2(r-1)}{mc_{\min}}$ time units). We can bound the resulting makespan as follows:

$$\begin{aligned} M &\leq 2\left(p\left(\frac{k+1}{mc_{\min}} + \frac{2(k-1)}{mc_{\min}}\right) + \frac{k+1}{mc_{\min}} + \frac{2(r-1)}{mc_{\min}}\right) \\ &= \frac{2}{mc_{\min}}\left(2\underbrace{(pk+r)}_{=m} + \underbrace{pk}_{\leq m} + \underbrace{k}_{<m} - p - 1\right) \\ &< \frac{8}{c_{\min}} \\ &\leq 8\textsc{Opt} \end{aligned}$$

Note that in this case we do not need the approximation guarantee of Scale-Fit, and in particular we do not need $c_0 = c_{\max}$, thus we can decrease the approximation guarantee by 1.

We now turn to the running time. Spread-Exchange is run p times for k packets, and once for r packets. Since p is roughly m/k and $r < m$, this results in an asymptotic complexity of $O(\frac{m}{k}(n \log n + nk) + n \log n + nm) = O(\frac{m}{k} n \log n + nm)$. In the worst case, the height k of the distribution tree is constant, so we get a worst-case complexity of $O(nm \log n)$. □

We conclude this section with an observation, relating our result to that of Mundinger et al. [18].

Remark 16 If the number of nodes is a power of two and capacities are homogeneous, i.e. $|N| = 2^\ell$ for some $\ell \in \mathbb{N}$ and $c_i = 1$ for all $i \in N$, the solution produced by Spread-Exchange is exactly the same as the one produced by the algorithm presented in Section 3 of [18]. (Note that we do not have to scale capacities in this case and thus also do not scale the sending rates in the end.) This also implies that in this special case Spread-Exchange yields an optimal solution.

4.2 An Algorithm for Many Packets

In this section, we devise a simple alternative algorithm, which we call Spread-Mirror-Cycle, that has a better performance guarantee than Spread-Exchange when the number of packets is large compared to the number of nodes. Our algorithm proceeds in three phases. In the first phase (Spread), we choose a certain subset of nodes, and send each packet to exactly one node from this subset. In the second phase (Mirror), we let these nodes *mirror* their packets to the remaining nodes, in such a way that the distribution of packets among the initial set of nodes is copied (possibly several times), and at the end of phase two every node owns at least one packet and each packet is owned by at least one node. In the final phase of the algorithm (Cycle), we perform a rather simple circular exchange of packets between nodes until all nodes possess the whole file. The main difference to the Spread-Exchange-algorithm, described in the previous section, is that Spread-Exchange uses Scale-Fit as a subroutine to provide each node with one packet. We use a different method for this task, that aims for a large variety of the distributed packets. This comes at the cost of a worse performance guarantee for the first two phases, but allows for more flexibility and hence a better performance of the exchange part our algorithm (Cycle). When the number of packets is large compared to the number of nodes, the exchange phase dominates the running time of the algorithm, and Spread-Mirror-Cycle has a better worst-case performance than Spread-Exchange.

We now describe Spread-Mirror-Cycle in detail. In contrast to Spread-Exchange, we do not round the capacities to powers of 2, but work on the original capacities. Let $k = \lceil n/m \rceil$. We partition the set of nodes into k mutually disjoint sets $N_1, \ldots, N_k$. The partition is chosen in such a way that the cardinalities differ by at most one, and the sets are indexed such that the cardinalities are non-decreasing. Formally, for $j \in \{1, \ldots, k\}$, let $n_j = |N_j|$ denote the cardinality of the j-th set and let $N_j = \{i_1^j, \ldots, i_{n_j}^j\}$. Then we have $|n_j - n_{j'}| \leq 1$ for all $j, j' \in \{1, \ldots, k\}$, and $n_j \leq n_{j'}$ for all $j, j' \in \{1 \ldots, k\}$ with $j \leq j'$. Note that by the choice of k, each partition contains more than $m/2$ nodes, but at most m, i.e. $m/2 < n_j \leq m$ for all j.

In the Spread phase of our algorithm, node 0 sends one packet to each of the nodes in N_1 as follows: at time 0, node 0 starts sending packet 1 to node i_1^1 at maximum rate $\min\{c_0, c_{i_1^1}\}$. After this transmission is completed, node 0 sends packet 2 to node i_2^1 at maximum rate $\min\{c_0, c_{i_2^1}\}$, and so on. This process continues

until node 0 has sent packet n_1 to node $i^1_{n_1}$. Then, if $m > n_1$, nodes $i^1_1, \ldots, i^1_{m-n_1}$ one after another receive a second packet from node 0 at full rate, until each node $i \in N_1$ owns at least one packet and each packet is owned by at least one node in N_1. For each node $j \in \{1, \ldots, n_1\}$, the transmission of a packet needs at most $\frac{1}{mc_{\min}}$ time units.

In the Mirror phase, all nodes in N_1 mirror themselves to a node in N_2. More formally, we form a matching between the nodes in N_1 and N_2. Assume for the moment that the sets are of equal size. Each node i sends its packets to its matched node i' at the full rate $\min\{c_i, c_{i'}\}$. As each node sends at most two packets, this process needs at most $\frac{2}{mc_{\min}}$ time units. We then continue iteratively mirroring the internal node structure of N_1 to all other subsets of nodes. That is, the nodes in N_1 send their packets to the nodes in N_3. At the same time, the nodes in N_2 send their packets to the nodes in N_4, and so on. After at most $\frac{2\lceil \log_2 k \rceil}{mc_{\min}}$ time units, the internal structure of N_1 is mirrored to all other sets $N_2, \ldots N_k$.

When the cardinalities of the two sets are different, i.e. nodes in set N_j mirror themselves to the nodes in set $N_{j'}$ with $n_{j'} = n_j + 1$, we slightly correct the mirroring process to take care of that fact. As $n_j = n_{j'} - 1 \leq m - 1$, there is a node $i^* \in N_j$ that owns two packets. We match this node to two different nodes in $N_{j'}$, and let it send one packet to each of them. In that fashion, we ensure that after the mirroring, all nodes in $N_{j'}$ possess at least one packet.

In the Cycle phase, we perform a simple circular exchange in all sets N_j, $j \in \{1, \ldots, k\}$. Recall that within each set N_j, each node owns at least one packet and each packet is present at exactly one node. The Cycle phase consists of $m-1$ rounds. In each round, node i^j_k sends one of its packets to node i^j_{k+1}, except for $i^j_{n_j}$, who sends to i^j_1. Each node sends each packet only once, in order of their reception. After $m-1$ rounds of such exchange, all nodes in N_j possess all packets. To see this, note that during the $m-1$ exchange rounds the order of the packets sent along the cycle is preserved. Hence, each node receives all other packets before it gets the packet it sent first. Thus, each node has all packets after $m-1$ rounds. Performing the exchange in all sets $N_1, \ldots, N_k$ in parallel, the final phase of our algorithm needs $\frac{m-1}{mc_{\min}}$ time units.

Summing up, we get the following result:

Theorem 17 Spread-Mirror-Cycle *yields a* $(2 + 2\lceil \log_2 \lceil n/m \rceil \rceil / m)$*-approximation in time* $O(nm)$.

Proof Adding up the time spent on the single phases, the total makespan of the solution computed by Spread-Mirror-Cycle is bounded by

$$\frac{1}{c_{\min}} + \frac{2\lceil \log_2 \lceil n/m \rceil \rceil}{mc_{\min}} + \frac{m-1}{mc_{\min}} \leq 2\textsc{Opt} + \frac{2\lceil \log_2 \lceil n/m \rceil \rceil}{m} \textsc{Opt} .$$

The partitioning of the set can be done in time $O(n)$, the sending rates for the Spread phase are computed in time $O(m)$. Then, there are $O(k) = O(n/m)$ Mirror steps, each of which can be computed in time $O(m)$. Finally, there is the Cycle phase, in which each node sends $m-1$ packets. Computing the sending rates of this phase thus takes time $O(nm)$, dominating the overall complexity of the algorithm. □

5 Conclusion

We studied the problem of distributing a file, divided into m packets, to all nodes of a communication network. In contrast to the prevailing assumption in the broadcasting literature known as the telephone model (and variants thereof), our model allows for flexible data transfers, that is, at any point in time each node that possesses a certain packet may send it with an arbitrary rate to other nodes that have not yet received this packet. For this quite general model, we provided a detailed study of various settings. For the simplest setting with homogeneous capacities and a single packet, we presented an efficient exact algorithm. Already for a single packet and heterogeneous capacities, the problem becomes strongly NP-hard. We thus turned to approximation algorithms and devised constant factor approximation algorithms for heterogeneous capacities, both for single and multiple packets. Some questions remain open, however, such as the case of different upload and download capacities per node. It would also be interesting whether the presented approximation factors can be improved, or whether inapproximability bounds can be shown.

References

1. E. Arkin, M. Bender, S. Fekete, J. Mitchell, and M. Skutella. The freeze-tag problem: How to wake up a swarm of robots. *Algorithmica*, 46(2):193–221, 2006.
2. E. Arkin, M. Bender, D. Ge, S. Hejoseph, and S. Mitchell. Improved approximation algorithms for the freeze-tag problem. In *In Proceedings of the 15th annual ACM symposium on Parallel algorithms and architectures*, pages 295–303, 2003.
3. M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical report, University of California at Berkeley, February 2009.
4. A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Message multicasting in heterogeneous networks. *SIAM J. Comput.*, 30(2):347–358, 2000.
5. A. Bar-Noy, S. Kipnis, and B. Schieber. Optimal multiple message broadcasting in telephone-like communication systems. *Discrete Applied Mathematics*, 100(1-2):1–15, 2000.
6. G. Ezovski, A. Tang, and L. Andrew. Minimizing average finish time in P2P networks. In *Proc. 28th IEEE Internat. Conf. on Computer Comm. (INFOCOM)*, pages 594–602, 2009.
7. B. Fan, J. C. S. Lui, and D.-M. Chiu. The Design Trade-Offs of BitTorrent-Like File Sharing Protocols. *IEEE/ACM Transactions on Networking*, 17:365–376, 2009.
8. M. Garey and D. Johnson. *Computers and Intractability*. W. H. Freeman, New York, NY, USA, 1979.
9. K.-S. Goetzmann, T. Harks, M. Klimm, and K. Miller. Optimal file distribution in peer-to-peer networks. In *Proc. 22nd Internat. Conf. on Algorithms and Computation*, pages 210–219, 2011.
10. S. T. Hedetniemi, S. M. Hedetniemi, and A. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18:129–134, 1998.
11. S. Khuller and Y.-A. Kim. Broadcasting in heterogeneous networks. *Algorithmica*, 48(1):1–21, 2007.
12. J. Könemann, A. Levin, and A. Sinha. Approximating the degree-bounded minimum diameter spanning tree problem. *Algorithmica*, 41(2):117–129, 2005.
13. R. Kumar and K. Ross. Peer assisted file distribution: The minimum distribution time. In *Proc. 1st IEEE Workshop on Hot Topics in Web Systems and Technologies*, pages 1–11, 2006.
14. O.-H. Kwon and K.-Y. Chwa. Multiple message broadcasting in communication networks. *Networks*, 26(4):253–261, 1995.
15. M. Mehyar, W. Gu, S. Low, M. Effros, and T. Ho. Optimal strategies for efficient peer-to-peer file sharing. In *Proc. IEEE Internat. Conf. Acoustics, Speech and Signal Process.*, volume 4, pages 1337–1340, 2007.
16. M. Middendorf. Minimum broadcast time is NP-complete for 3-regular planar graphs and deadline 2. *Inf. Process. Lett.*, 46(6):281–287, 1993.
17. K. Miller and A. Wolisz. Transport optimization in peer-to-peer networks. In *Proc. 19th Internat. Conf. on Parallel, Distributed and Network-Based Process.*, pages 567–573, 2011.
18. J. Mundinger, R. Weber, and G. Weiss. Optimal scheduling of peer-to-peer file dissemination. *J. of Scheduling*, 11:105–120, 2008.
19. D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *Proc. ACM Conf. Applications, Technologies, Architectures, and Protocols for Comput. Comm. (SIGCOMM)*, pages 367–378, 2004.
20. R. Ravi. Rapid rumor ramification: Approximating the minimum broadcast time. In *Proc. 35th Annual IEEE Sympos. Foundations Comput. Sci.*, pages 202–213, 1994.