Goetzmann, Kai-Simon; Harks, Tobias; Klimm, Max



# Broadcasting a file in a communication network

Journal of Scheduling

**Provided in Cooperation with:**
Springer Nature

*Suggested Citation:* Goetzmann, Kai-Simon; Harks, Tobias; Klimm, Max (2020) : Broadcasting a file in a communication network, Journal of Scheduling, ISSN 1099-1425, Springer US, New York, NY, Vol. 23, Iss. 2, pp. 211-232, https://doi.org/10.1007/s10951-020-00643-w

Journal of Scheduling
https://doi.org/10.1007/s10951-020-00643-w

# Broadcasting a file in a communication network

**Kai-Simon Goetzmann**[1] · **Tobias Harks**[3] · **Max Klimm**[2]



**Abstract**

We study the problem of distributing a file, initially located at a server, among a set of $n$ nodes. The file is divided into $m \geq 1$ equally sized packets. After downloading a packet, nodes can upload it to other nodes, possibly to multiple 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 Sched 11:105–120, 2008. https://doi.org/10.1007/s10951-007-0017-9) 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 5 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 (Armbrust et al. 2009). 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 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),

The results of Section 3 of this paper appeared as an extended abstract in Proceedings of the 22nd International Symposium on Algorithms and Computation (Goetzmann et al. 2011).
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 (MI-8).

✉ Max Klimm
max.klimm@hu-berlin.de

Tobias Harks
tobias.harks@math.uni-augsburg.de

[1] Institut für Mathematik, Technische Universität Berlin, Straße des 17. Juni 136, 10623 Berlin, Germany

[2] School of Business and Economics, Humboldt-Universität zu Berlin, Spandauer Straße 1, 10178 Berlin, Germany

[3] Institut für Mathematik, Universität Augsburg, 86135 Augsburg, Germany

Published online: 11 February 2020

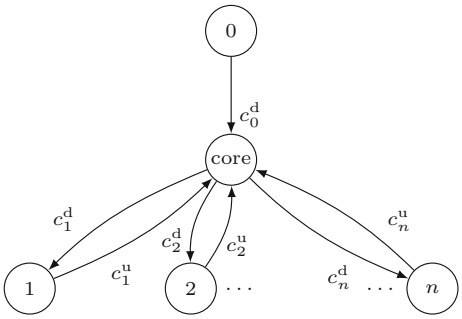



**Fig. 1** Graphical representation of the file distribution problem we consider

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* $c_i^{\mathrm{u}}$, and the sum of the sending rates of $i$ may not exceed $c_i^{\mathrm{u}}$. Second, every node has a *download capacity* $c_i^{\mathrm{d}}$.

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 Fig. 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 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 a large body of work on the (minimum time) broadcasting problem, multicast problem, and gossiping problem, see Hedetniemi et al. (1998) 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 (1994) 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. (2000a) 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 1979), even for 3-regular planar graphs (cf. Middendorf 1993).

Khuller and Kim (2007) studied the problem of broadcasting in heterogeneous networks [cf. Bar-Noy et al. (2000a) 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 and Kim (2007)], 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 and Kim (2007) 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. (2008) 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 (1995) and Bar-Noy et al. (2000b), 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 fluid flow model, for which a relatively easy closed-form expression of the minimum completion time can be derived, see, e.g., Ezovski et al. (2009), Kumar et al. (2006), and Mehyar et al. (2007). Qiu and Srikant (2004) studied a related model with stochastic arrival of peers. Fan et al. (2009) considered a finite number of parts, but assumed 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. (2003) 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 $\Theta\left(\sqrt{\log n}\right)$-approximation, where $n$ is the number of robots. Könemann et al. (2005) devised a $\mathcal{O}\left(\sqrt{\log n}\right)$-approximation for the weighted case. Arkin et al. (2006) 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 Sect. 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. (2008), or in the broadcasting literature (Kwon and Chwa 1995; Bar-Noy et al. 2000b), 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 Sect. 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 the first phase, 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 4-approximation of the optimal makespan. For smaller server capacities, a slight variation of the algorithm gives a 5-approximation.

Motivated by real-world instances where 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. This allows to perform a faster exchange procedure in the second phase. 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 $n \lessapprox m2^{(3/2)\,m}$.

## 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 $N = \{0, \ldots, n\}$ is the set of nodes, $m \in \mathbb{N}_{\geq 1}$ is the number of packets, $\mathbf{c}^{\mathrm{d}} = (c_0^{\mathrm{d}}, \ldots, c_n^{\mathrm{d}}) \in \mathbb{Q}_{\geq 0}^{n+1}$ is the vector of download capacities from the core network and $\mathbf{c}^{\mathrm{u}} = (c_0^{\mathrm{u}}, \ldots, c_n^{\mathrm{u}}) \in \mathbb{Q}_{\geq 0}^{n+1}$ is the vector of upload capacities to the core network. Let $[m] = \{1, 2, \ldots, m\}$ be the set of packets. We will identify the server with node 0 and assume that only the server initially owns the file. See Fig. 1 for an illustration. The file is of unit size; the size of each packet is thus $1/m$. A feasible solution $S = \left(s_{i,j}^{(k)}\right)_{i,j \in N, k \in [m]}$ 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 $p^{(k)}(i) \neq i$:

$$s_{\ell,i}^{(k)}(t) = 0 \text{ for all } \ell \neq p^{(k)}(i) \text{ and all } t \in \mathbb{R}_{\geq 0}.$$

In addition, only nodes that possess a packet $k$ can send that packet with a positive rate:

$$s_{i,j}^{(k)}(t) = 0 \text{ for all } i, j \in N \setminus \{0\}, k \in [m], t \in \mathbb{R}_{\geq 0} \text{ with } \int_0^t \sum_{\ell \in N} s_{\ell,i}^{(k)}(\tau)\,\mathrm{d}\tau < \frac{1}{m}.$$

Finally, the sending rates have to obey download and upload capacity constraints:

$$\sum_{k \in [m]} \sum_{j \in N} s_{i,j}^{(k)}(t) \leq c_i^{\mathrm{u}} \text{ for all } i \in N, t \in \mathbb{R}_{\geq 0}, \text{ and}$$
$$\sum_{k \in [m]} s_{p^{(k)}(j),j}^{(k)}(t) \leq c_j^{\mathrm{d}} \text{ for all } j \in N, t \in \mathbb{R}_{\geq 0}.$$

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 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 $M(S) = \max_{i \in N \setminus \{0\}} C_i$. 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 $\rho$-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 $\mathrm{OPT}(I)$ by more than a factor of $\rho$, i.e., $c(S) \leq \rho\,\mathrm{OPT}(I)$.

There are some particularities of the minimum time broadcasting problem regarding the polynomial running time of algorithms. In general, a solution may consist 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 Miller and Wolisz (2011, 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

**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**

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 later 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 $\mathcal{O}(n \log n)$-algorithm that approximates the optimal makespan by a factor of $1 + 2\sqrt{2}$.

### 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 $\mathcal{O}(n)$. 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

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

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:

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

$$s'_{0,i}(t) = \begin{cases} 0 & \text{for } t \in [0,1), \\ \frac{x_i^*(C_1^*)}{C_1^* - 1} & \text{for } t \in [1, C_1^*), \\ s^*_{0,i}(t) & \text{for } t \geq C_1^*, \end{cases}$$

for all $i \in N \setminus \{1\}$ with $p(i) = 0$,

$$s'_{i,j}(t) = s^*_{i,j}(t)$$

for all other $i, j \in N$ and $t \in \mathbb{R}_{\geq 0}$. 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 Miller and Wolisz (2011, 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 $t \in [1, C_1^*)$,

$$\begin{aligned} &\sum\nolimits_{i \in N \setminus \{1\} : p(i) = 0} s'_{0,i}(t) \\ &= \frac{1}{C_1^* - 1} \sum\nolimits_{i \in N \setminus \{1\} : p(i) = 0} x_i^*(C_1^*) \\ &= \frac{C_1^*}{C_1^* - 1} \sum\nolimits_{i \in N \setminus \{1\} : p(i) = 0} s^*_{0,i}(0) \\ &= \frac{1}{1 - \frac{1}{C_1^*}} \Bigg( \underbrace{\sum\nolimits_{i \in N} s^*_{0,i}(0)}_{\leq 1} - s^*_{0,1}(0) \Bigg) \\ &\leq \frac{1}{1 - s^*_{0,1}(0)} \Big( 1 - s^*_{0,1}(0) \Big) \\ &= 1. \end{aligned}$$

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 the 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 $c_i = 1$ for all $i = 1, \ldots, n$, there always exists an optimal solution that uses* fair sharing.

***Proof*** For an arbitrary instance $I$ 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 let $\ell_1, \ell_2, \ldots, \in \{1, \ldots, k\}$ be such that

$$\begin{aligned} C^*_1 &= \ldots = C^*_{\ell_1} \\ &< C^*_{\ell_1+1} = \cdots = C^*_{\ell_1+\ell_2} < C^*_{\ell_1+\ell_2+1} \leq \ldots \leq C^*_k. \end{aligned}$$

Let $N_1 = \{1, \ldots, \ell_1\}$. 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.,

$$m_i = \max\left\{C^*_j - C^*_i : j \in N,\, p^r(j) = i \text{ for some } r \in \mathbb{N}\right\},$$

where $p^r(j)$ denotes the $r$-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 $M_1 = \max_{i \in N_1} m_i$.

In the following, we show how to modify the original solution $s^*$ to obtain a solution $s'$ that uses fair sharing and has not a larger makespan. All primed variables $s'_{i,j}$, $C'$, $x'$, $\ell'_1$ and $N'_1$ refer to variables of the modified solution. Whenever $s'_{i,j}(t)$ is not specified for some $i$, $j$, $t$, it is equal to the original value $s^*_{i,j}(t)$. The modification of $s^*$ to $s'$ proceeds iteratively in steps. In each step, one of four modifications is made. Specifically, if the server capacity is not smaller than $k$, all nodes $\{1, \ldots, k\}$ are served by the server at equal rate (Case 1). If $k > c_0$ and $M_1 \leq m_{i_0}$ for some $i_0 \in \{\ell_1 + 1, \ldots, k\}$, a single node is added to $N_1$ and all nodes in $N_1$ are served with equal rate while maintaining the original rates for all nodes not in $N_1$ (Case 2). If $\ell_1 \geq c_0$ and $M_1 > m_i$ for all $i \in \{\ell_1 + 1, \ldots, k\}$, the server serves all nodes in $N_1$ at full capacity and with fair sharing; the serving of the other nodes is delayed (Case 3). Finally, if $\ell_1 < c_0 < k$ and $M_1 > m_i$ for all $i \in \{\ell_1 + 1, \ldots, k\}$, we increase $N_1$ to size $c_0$ and serve the nodes in $N_1$ at full capacity with fair sharing; the serving of all other nodes is delayed (Case 4).

Application of Case 1 immediately leaves us with a solution that uses fair sharing. Application of Case 3 and Case 4 leaves us with a solution $s'$ with completion times

$$\begin{aligned} C'_1 &= \ldots = C'_{\ell'_1} \\ &< C'_{\ell'_1+1} = \cdots = C'_{\ell'_1+\ell'_2} < C_{\ell'_1+\ell'_2+1} \leq \ldots \leq C'_k, \end{aligned}$$

where the nodes in $N'_1 = \{1, \ldots, \ell'_1\}$ are served in time $[0, C'_1)$ with fair sharing and nodes $i \in \{\ell'_1 + 1, \ldots, k\}$ are served by the server after $C'_1$. We then fix the part of the solution related to the nodes in $N'_1$ (and the nodes served by these nodes) and apply the same modifications for the nodes in $N'_2 = \{\ell'_1 + 1, \ldots, \ell'_1 + \ell'_2\}$. Finally, after application of Case 2, we have increased the size of $N_1$ by one. Since the number of nodes is finite, after a finite number of applications of Case 2, one of the other three cases is applied.

We proceed to show that the application of the four cases yields a feasible solution and does not increase the makespan.

**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 $i_0 \in \{\ell_1 + 1, \ldots, k\}$.

We add one node to $N_1$, serving nodes $1, \ldots, \ell_1 + 1$ with equal rates in the time interval $\left[0, C^*_{\ell_1+1}\right)$:

$$\forall\, i \in 1, \ldots, \ell_1 + 1 : \quad s'_{0,i}(t) = \begin{cases} \sum_{j=1}^{\ell_1+1} \frac{s^*_{0,j}(t)}{\ell_1+1} & \text{for all } t \in \left[0, C^*_{\ell_1+1}\right), \\ 0 & \text{otherwise.} \end{cases}$$

Then nodes $1, \ldots, \ell_1+1$ all complete their download at time $C^*_{\ell_1+1}$ since

$$x_i'(C^*_{\ell_1+1}) = \int_0^{C^*_{\ell_1+1}} s'_{0,i}(\tau)\ \mathrm{d}\tau = \frac{1}{\ell_1+1}\sum_{j=1}^{\ell_1+1}\underbrace{\int_0^{C^*_{\ell_1+1}} s^*_{0,j}(\tau)\ \mathrm{d}\tau}_{=1} = 1$$

for all $i = 1, \ldots, \ell_1+1$. The download capacities of all nodes are satisfied since at any point $t$ in time

$$s'_{0,i}(t) = \frac{1}{\ell_1+1}\sum_{j=1}^{\ell_1+1} s^*_{0,j}(t) \le \max_{j=1,\ldots,\ell_1+1} s^*_{0,j}(t) \le 1.$$

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.

We proceed to argue that this modification does not increase the makespan. First, since $M_1 \le m_{i_0}$, for all $i \in N_1$ we have

$$C'_i + m_i = C^*_{k'+1} + m_i \le C^*_{i_0} + m_{i_0} \le M^*,$$

so delaying the completion of nodes in $N_1$ does not influence the makespan. Furthermore, in the modified solution $s'$, all nodes $i \in \{\ell_1+2, \ldots, k\}$ are served by the server in the same way as in solution $s^*$ so that their completion time $C_i$ as well as the completion times of the nodes served by them does not change.

We now turn to the cases where $M_1 > m_i$ for all $i = \ell_1+1, \ldots, k$. Note that by Lemma 1, we know that $m_i \in \mathbb{N}$ for all $i$; hence, we can assume $M_1 \ge m_i + 1$ for all $i > \ell_1$ in the subsequent cases.

**Case 3:** $\ell_1 \ge c_0$ and $M_1 \ge m_i + 1$ for all $i > \ell_1$.
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 $t_1 = \ell_1/c_0$, and set

$$\forall\, i \in N_1: \quad s'_{0,i}(t) = \begin{cases} \frac{c_0}{\ell_1} & \text{for all } t \in [0, t_1) \\ 0 & \text{otherwise.} \end{cases}$$
$$\forall\, i = \ell_1+1, \ldots, k: \quad s'_{0,i}(t) = 0 \quad \text{for all } t \in [0, t_1).$$

It is obvious that in $[0, t_1)$ all capacities are obeyed and $C'_i = t_1 \le C^*_i$ 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 $\ell_1+1, \ldots, k$ 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., $x'_i(t_2) = x^*_i(t_2)$:

$$\forall\, i = \ell_1+1, \ldots, k: \quad s'_{0,i}(t) = \frac{x^*_i(t_2)}{t_2 - t_1} \text{ for all } t \in [t_1, t_2).$$

Since $x^*_i(t_2) \le 1$ and $t_2 - t_1 \ge 1$, this solution 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:

$$\begin{aligned}\int_0^{t_2}\sum_{i\in N} s'_{0,i}(\tau)\,\mathrm{d}\tau &= \sum_{i=1}^{\ell_1}\int_0^{t_1}\frac{c_0}{\ell_1}\,\mathrm{d}\tau + \sum_{i=\ell_1+1}^{k}\int_{t_1}^{t_2}\frac{x^*_i(t_2)}{t_2-t_1}\,\mathrm{d}\tau \\ &= \sum_{i=1}^{k'}\underbrace{t_1\cdot\frac{c_0}{\ell_1}}_{=1=x^*_i(t_2)} + \sum_{i=\ell_1+1}^{k} x^*_i(t_2) \\ &= \sum_{i=1}^{k} x^*_i(t_2),\end{aligned}$$

and the overall rate the server sends at is $c_0$ at the beginning and constant afterward, hence it is at most $c_0$ at all times. From time $t_2$ on, in the modified solution $s'$, all nodes are served as in the original solution $s^*$.

Nodes in $N_1$ and nodes with $C^*_i \ge t_2$ complete their download no later than in the original solution. If there is a node $i \in \{\ell_1+1, \ldots, k\}$ 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 $M_1 \ge m_i + 1$ we obtain that

$$C'_i + m_i = t_1 + 1 + m_i \le C^*_1 + M_1 \le M^*,$$

therefore the makespan is not increased.

**Case 4:** $\ell_1 < c_0$ and $M_1 \ge m_i + 1$ for all $i > \ell_1$.
In this case, we add nodes $\ell_1+1, \ldots, c_0$ to $N_1$ and serve these nodes jointly at full capacity in the time interval $[0, 1)$:

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

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

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

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 $c_i = 1$ for all $i = 1, \ldots, n$, 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 served simultaneously until they are completed.*

***Proof*** For an 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 $t_j = \sum_{\ell=1}^{j} \max\{1, \frac{|N_\ell|}{c_0}\}$ 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.,

$$\tilde{N}_j = \left\{i \in N : p^r(i) = i' \text{ for some } i' \in N_j, r \in \mathbb{N}\right\},$$
$$M_j = \max_{i \in \tilde{N}_j} C_i^* - t_j, \tag{1}$$

where again $p^r(j)$ denotes the $r$-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 $M_1 \leq M_2 + 1$, we remove $|N_2| - c_0$ nodes from $|N_2|$, add them to $N_1$ and let them be served by the server jointly with the other 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. □

**Algorithm 2:** EXTENDED GREEDY for integral server capacity

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

$h := \lceil \log_2(\frac{n}{c_0} + 1) \rceil$;
**if** $n \geq c_0(2^h - 1 + 2^{h-1})$ **then**
  **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**
  **Choose** $N_1 \subseteq N$ with $|N_1| = \lceil \frac{n - c_0(2^{h-1}-1)}{2^{h-1}} \rceil$;
  $k := h$ ;
**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** $N_{t+2} \subseteq N$, $|N_{t+2}| = c_0$ of unserved nodes;
  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**

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 $|N_j| = c_0$ for all $j \geq 2$, 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 $c_i = 1$ for all $i = 1, \ldots, n$, Algorithm 2 yields an optimal solution. The optimal makespan is*

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

*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 $c_0 \leq |N_1| < 2c_0$. In the argumentation below, we therefore restrict to solutions of this structure.

The integer $h$ is chosen such that $c_0(2^h - 1) \leq n < c_0(2^{h+1} - 1)$. We first consider the case where $n \geq c_0(2^h - 1 + 2^{h-1})$. 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 $|N_1| \cdot 2^{h-1} + c_0(2^{h-1} - 1) < c_0(2^h + 2^{h-1} - 1) \leq n$, so not all nodes can be served, contradiction! Therefore in an optimal solution it must hold that $k = h + 1$ and $|N_1| \geq c_0$, 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 $j \geq 2$) has to be chosen such that $|\tilde{N}_1| = n - c_0(2^{h-1} - 1) \leq |N_1| \cdot 2^{h-1}$, i.e.,

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

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

$$z_i(t_1, t_2) = \int_{\max\{t_1, C_i\}}^{t_2} \left( c_i - \sum_{j \in N} s_{i,j}(\tau) \right) \mathrm{d}\tau.$$

For $t \geq 0$, 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 $0 \leq t_1 \leq t_2$.

**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. *For all nodes $i \in N$ and all times $0 \leq t_1 < t_2$:*

$$\begin{aligned} &u_i(t_1, t_2) + z_i(t_1, t_2) \\ &\quad \leq \max\left\{0, \left(t_2 - t_1 - \frac{1 - x_i(t_1)}{c_i}\right) c_i\right\} \\ &\quad = \max\left\{0, (t_2 - t_1)\, c_i - 1 + x_i(t_1)\right\} \end{aligned}$$

2. *$X\left(\frac{k}{c}\right) + Z\left(\frac{k}{c}\right) \leq 2^k$ for all $k \in \mathbb{N}$. This inequality is strict if there is $i \in N$ with $c_i < c$ and $0 < C_i \leq \frac{k}{c}$.*

***Proof*** To see 1., note that for any node $i$ with $C_i \leq t_1$, 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 $u_i(t_1, t_2) + z_i(t_1, t_2) \leq (t_2 - t_1)\, c_i$. 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 $X\left(\frac{k-1}{c}\right)+Z\left(\frac{k-1}{c}\right) \leq 2^{k-1}$ and that this inequality is strict if there is $i \in N$ with $c_i < c$ and $0 < C_i \leq \frac{k-1}{c}$. Using 1., we obtain

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

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

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

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 $X\left(\frac{k-1}{c}\right) < 2^{k-1}$ and, by (3), $X\left(\frac{k}{c}\right) < 2^k$. If, on the other hand, $C_i \in \left(\frac{k-1}{c}, \frac{k}{c}\right]$, then the inequality (2) is satisfied strictly and we obtain $X\left(\frac{k}{c}\right) < 2^k$ 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 $\mu$ 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 (1979). We first construct a modified 3- PARTITION instance in which $\mu$ 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 $\mu'$ 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 $c_{i_k} = \frac{k_i}{\ell k_i - 1}$ 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 $\frac{\ell}{B} + \ell$ 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. Afterward all element nodes are served. More formally, in the first $\frac{1}{B}$ time units subset node $s_0$ sends the file to the first subset node $s_1$ with a rate of $s_{s_0,s_1}(t) = B$ for all $t \in [0, \frac{1}{B})$. 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 $s_{s_0,s_2}(t) = s_{s_1,s_3}(t) = B$ for all $t \in \left[\frac{1}{B}, \frac{2}{B}\right)$. Continuing in this fashion, at time $\ell/B$ all $\mu'$ 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 $\ell$ 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 $s_{s_j,i_0}(t) = k_i$ for all $t \in \left[\frac{\ell}{B} + \frac{1}{k_i}\right)$. 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 $\frac{\ell}{B} + \frac{1}{k_i}$. 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 $s_{i_0,i_j}(t) = \frac{k_i}{\ell k_i - 1}$ for all $t \in \left[\frac{\ell}{B} + \frac{1}{k_i}, \frac{\ell}{B} + \ell\right)$. The remaining $\ell k_i - 1$ element nodes $i_j$, $j \in \{\ell k_i, 2\ell k_i - 2\}$, are served by the subset node $s_j$ with rate $s_{s_j,i_j}(t) = \frac{k_i}{\ell k_i - 1}$ for all $t \in \left[\frac{\ell}{B} + \frac{1}{k_i}, \frac{\ell}{B} + \ell\right)$. The download of the file by the element nodes starts at time $\frac{\ell}{B} + \frac{1}{k_i}$ and needs additional $\frac{\ell k_i - 1}{k_i}$ time units, resulting in a total makespan of $\frac{\ell}{B} + \ell$.

It remains to be shown that the optimal makespan of the broadcasting instance $I$ is strictly larger than $\frac{\ell}{B} + \ell$ 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 $\frac{\ell k_i - 1}{k_i} = \ell - \frac{1}{k_i}$ 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 $\bar{S}$ of the partial instance $\bar{I}$ that consists only of subset nodes and the master element nodes and has accumulated enough idleness until time $\frac{\ell}{B} + \ell$, i.e.,

$$Z\left(\frac{\ell}{B} + \ell\right) \geq \sum_{i \in P'} (2\ell k_i - 2).$$

Otherwise the idleness of subset nodes and master element nodes does not suffice to serve all element nodes until time $\frac{\ell}{B} + \ell$ in the original instance $S$. We proceed to show that for the partial instance, no such partial solution exists.

Let us first assume that $x_{s_j}\left(\frac{\ell}{B}\right) = 1$ for every subset node $s_j$. Using Lemma 5, the inequality $X\left(\frac{\ell}{B}\right) + Z\left(\frac{\ell}{B}\right) \leq \mu'$ holds, thus $Z\left(\frac{\ell}{B}\right) = 0$ and $x_{i_0}\left(\frac{\ell}{B}\right) = 0$ for every master element node $i_0$. We calculate

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

For the completion time $C_{i_0}$ of a master element node $i_0$, we obtain the inequality $C_{i_0} \geq \frac{\ell}{B} + \frac{1}{k_i}$. We claim that there is at least one master element node $i_0'$ for which this inequality is strict. For a contradiction, suppose that $C_{i_0} = \frac{\ell}{B} + \frac{1}{k_i}$ 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 $C_{i_0'} > \frac{\ell}{B} + \frac{1}{k_i}$, establishing that

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

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

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

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

**First case:** At least one of the master element nodes has finished the download at time $\frac{\ell}{B}$.
Then, inequality (5) is strict and we obtain

$$\begin{aligned} Z\left(\tfrac{\ell}{B} + \ell\right) < \sum_{k_i \in P'} (2\ell k_i - 1) - \sum_{j=0}^{\mu'-1} \left(1 - x_{s_j}\left(\tfrac{\ell}{B}\right)\right) \\ - \sum_{k_i \in P'} \left(1 - x_{i_0}\left(\tfrac{\ell}{B}\right)\right) \end{aligned}$$

$$= \sum_{k_i \in P'} (2\ell k_i - 1) + \underbrace{X\left(\tfrac{\ell}{B}\right)}_{\leq \mu'} - \mu' - |P'|$$
$$\leq \sum_{k_i \in P'} (2\ell k_i - 2).$$

**Second case:** All master element nodes and at least one of the subset nodes still have not finished the downloads at time $\ell/B$.
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 $Z(\frac{\ell}{B} + \ell) < \sum_{k_i \in P'}(2\ell k_i - 2)$. □

### 3.2.2 Approximation

In this section, we devise an algorithm that runs in time $\mathcal{O}(n \log n)$ 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 summand of $M^*$ in the approximation arises as possibly we first have to transfer the file to the node having the largest capacity which takes at most $M^*$ time units.

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 the Capacity Expansion Lemma (Lemma 5). For *any* solution, node $i$'s contribution to the upload in time interval $[t_1, t_2]$ is bounded by $u_i(t_1, t_2) \leq \max\{0, c_i\,(t_2 - t_1) - 1 + x_i(t_1)\} - z_i(t_1, t_2)$ where $z_i(t_1, t_2)$ is the total idleness of node $i$ in the time interval $[t_1, t_2]$. 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 $\lambda$-*augmented* solution, if at any point in time the original capacities are not exceeded by more than a factor of $\lambda$, 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 $\sqrt{2}$.

We now explain the scaling of the capacities during the course of the algorithm, see also Algorithm 3 for a more formal description. The algorithm maintains a queue $Q$ of capacity release events

$$(t, i, c) \in \mathbb{R}_{\geq 0} \times N \times \mathbb{R}_{\geq 0}$$

meaning that at time $t$ a capacity of $c$ of node $i$ is available for uploads. At the start of the algorithm only the server is available for uploads so $Q$ has only a single entry $(0, 0, c_0)$.

Order the nodes non-increasingly by capacities such that $c_1 \geq c_2 \geq \cdots \geq c_n$ and consider a point in time $t$, where a node $i$ becomes available with its full upload capacity $c_i$ and suppose that nodes $1, \ldots, k-1$ have already received a fraction of the file. We now determine a number $k' \geq k$ such that nodes $k, \ldots, k'$ start receiving the file from node $i$. The number $k'$ is chosen such that either nodes $k, \ldots, k'$ receive the file at their full capacity and the upload factor of node $i$ is violated by a factor $\beta \in [1, \sqrt{2})$ or node $i$ sends the file at full capacity and the download capacity of all nodes $k, \ldots, k'$ is violated by a factor $\alpha \in [1, \sqrt{2})$. The next lemma shows that such $k'$ always exists (provided that there are enough nodes left).

**Lemma 7** *Let $c_1 \geq c_2 \geq \cdots \geq c_n$ and $c \geq \frac{c_1}{\sqrt{2}}$. If $\sum_{j=1}^{n} c_j > c$, then there is $k \leq n$ such that*

$$\frac{c}{\sqrt{2}} \leq \sum_{j=1}^{k} c_j \leq \sqrt{2}c.$$

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

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

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

During the course of the algorithm, nodes are served in order of non-increasing capacity. Since the capacities of the uploaders are inflated by a factor of at most $\sqrt{2}$, the released capacity $c$ satisfies $c \geq \frac{c_k}{\sqrt{2}}$, where $k$ is the smallest index of a node not served yet. Hence, we can apply Lemma 7 for the nodes $c_k \geq c_{k+1} \geq \cdots \geq c_n$ and derive the existence of $k' \geq k$ such that

$$\frac{c}{\sqrt{2}} \leq \sum_{j=k}^{k'} c_j \leq \sqrt{2}c.$$

The capacities of nodes $k, \ldots, k'$ are increased by a factor of

$$\alpha = \max\left\{1, \frac{c_i}{\sum_{j=k}^{k'} c_j}\right\}$$

and that of node $i$ by a factor of

$$\beta = \sum_{j=k}^{k'} \alpha \frac{c_j}{c_i},$$

see lines 13 and 15 in Algorithm 3.

For the instance with such augmented capacities, the schedule is now constructed as follows. At time $t$, nodes $j \in \{k, \ldots, k'\}$ start downloading from node $i$ with their full (possibly augmented) capacity of $\alpha c_j$ (line 20). Node $j \in \{k, \ldots, k'\}$ thus finishes the download at time $\tilde{C}_j = t + \frac{1}{\alpha c_j}$, see line 18. At that time, both the new upload capacity of node $j$ as well as the fraction of the upload capacity of node $i$ that was devoted to uploading the file to node $j$ become available for further uploads. To avoid double augmentation of capacities during the course of the algorithm, we have to rescale both to their original level. This is done in line 21. Here both the event that the (non-augmented) capacity $c_j$ of node $j$ is available and that a fraction $\alpha \frac{c_j}{\beta}$ of the capacity of node $i$ is available for further uploads is added to the event queue. Note that if $c_k > c_{k+1}$ at the time $\tilde{C}_k$ only some fraction of the upload capacity of node $i$ becomes available since nodes $k+1, \ldots, k'$ are still downloading from node $i$. The algorithm reassigns this fraction of the upload capacity at time $\tilde{C}_k$ without waiting for nodes $k+1, \ldots, k'$ to avoid idle times.

The algorithm proceeds by taking the next capacity release event out of the event queue. 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 $\sqrt{2}$.

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

***Proof*** When new downloads from a node $i$ are assigned, the download capacities are increased by the factor

$$\alpha = \max\left\{1, \frac{c}{\sum_{j=k}^{k'} c_j}\right\},$$

see line 13. By the choice of $k'$ in line 12, $\alpha \leq \sqrt{2}$ so that no download capacity is increased by a factor larger than $\sqrt{2}$. We prove that also the upload capacities are never exceeded by a factor larger than $\sqrt{2}$ by two inductive claims. First, we claim that after a node $i$ started the download of the file at time $\tilde{A}_i$, we have that the capacities of all further capacity release events for $i$ sum up to $c_i$ (as long as there are still unserved nodes), i.e.,

$$\sum_{(t',i',c') \in Q : i' = i} c' = c_i.$$

**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 $I$

1 **sort** nodes non-increasingly with respect to their capacities, i.e., $c_1 \geq c_2 \geq \cdots \geq c_n$ ;
// server has unused upload capacity $c_0$ at time 0
// add this capacity release to event queue
2 $Q := \{(0, 0, c_0)\}$;
// node with largest capacity not served yet
3 $k := 1$ ;
// largest augmentation factor so far
4 $b := 1$ ;
5 **while** $k \leq n$ **do**
// take next event from queue
6 $(t, i, c) := \arg\min_{(t',i',c') \in Q} t'$ ;
7 $Q := Q \setminus \{(t, i, c)\}$;
8 **if** $\sum_{j=k}^{n} c_j \leq \sqrt{2}c$ **then**
// $i$ serves all remaining nodes
9 $k' := n$;
// augmentation factor of downloader
10 $\alpha := 1$ ;
11 **else**
12 **choose** $k'$ such that $\frac{c}{\sqrt{2}} \leq \sum_{j=k}^{k'} c_j \leq \sqrt{2}c$ ;
// augmentation factor of downloader
13 $\alpha := \max\{1, c \,/\, \sum_{j=k}^{k'} c_j\}$ ;
14 **end**
// augmentation factor of uploader
15 $\beta := \sum_{j=k}^{k'} \alpha \frac{c_j}{c}$ ;
// update largest augmentation factor
16 $b := \max\{b, \alpha, \beta\}$ ;
// time when $j$ starts downloading
17 $\tilde{A}_j := t$ for all $j \in \{k, \ldots, k'\}$ ;
// time when $j$ finishes downloading
18 $\tilde{C}_j := t + \frac{1}{\alpha c_j}$ for all $j \in \{k, \ldots, k'\}$ ;
// parent of nodes $k, \ldots, k'$ is $j$
19 $p(j) := i$ for all $j \in \{k, \ldots, k'\}$ ;
// sending rate from $i$ to $j$
20 $\tilde{s}_{i,j}(\tau) := \alpha c_j$ for all $j \in \{k, \ldots, k'\}, \tau \in [\tilde{A}_j, \tilde{C}_j)$ ;
// add rescaled capacity release to queue
21 $Q := Q \cup \bigcup_{j=k}^{k'} \{(\tilde{C}_j, j, c_j) \cup (\tilde{C}_j, i, \alpha \frac{c_j}{\beta})\}$ ;
22 $k := k' + 1$;
23 **end**
24 **forall the** $i \in N \setminus \{0\}$ **do**
// Rescale $\tilde{S}$
25 $A_i := \tilde{A}_i b$ ;
26 $C_i := \tilde{C}_i b$ ;
27 $s_{p(i),i}(t) := \frac{\tilde{s}_{p(i),i}(\tilde{A}_i)}{b}$ for all $t \in [A_i, C_i)$ ;
28 **end**

This is obviously true for all times $t \in [\tilde{A}_i, \tilde{C}_i)$ when the only event in $Q$ is $(\tilde{C}_i, i, c_i)$. Next suppose that the claim is correct for all times until a capacity release event $(t, i, c)$ for $i$ is taken from the queue $Q$. Then, the capacity of $c$ is used to serve nodes $k, \ldots, k'$ where the download capacity of the downloaders is increased by the factor $\alpha = \max\{1, c \,/\, \sum_{j=k}^{k'} c_j\}$ and the upload capacity of $i$ is increased by the factor $\beta = \sum_{j=k}^{k'} \alpha \frac{c_j}{c}$. Finally, the capacity release events $\bigcup_{j=k}^{k'} (\tilde{C}_j, i, \alpha \frac{c_j}{\beta})$ are added for $i$. We have

$$\sum_{j=k}^{k'} \alpha \frac{c_j}{\beta} = \sum_{j=k}^{k'} \frac{c_j}{\sum_{\ell=k}^{k'} \frac{c_\ell}{c}} = c,$$

by the choice of $\beta$. This proves the claim.

Next, we observe that after node $i$ finished the download of the file at time $\tilde{C}_i$ and as long as there are still unserved nodes, every capacity release event $(t, i, \alpha \frac{c_j}{\beta})$ for $i$ corresponds to a an upload of node $i$ to some other node $j$ at rate $\alpha c_j$ and vice versa.

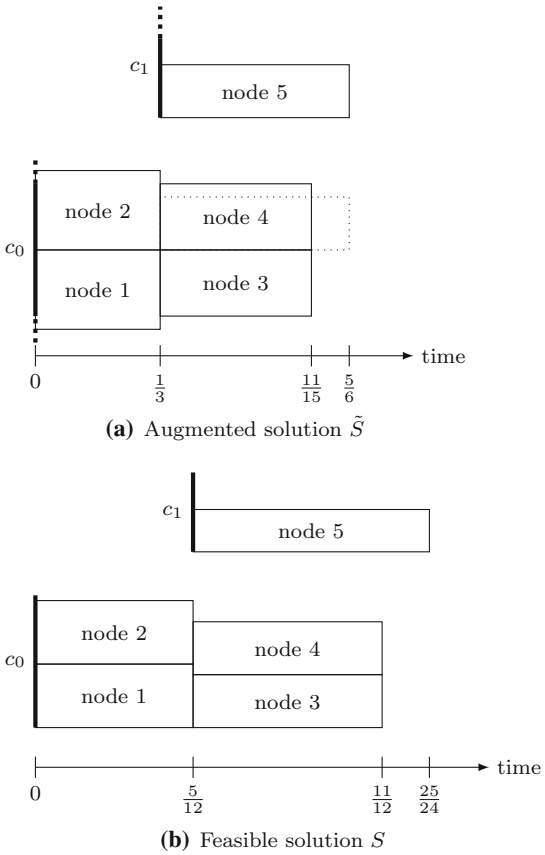
To sum up, for each node $i$, at every time $t \geq \tilde{C}_j$ all active uploads at rates $\alpha_j c_j$ for some $\alpha \in [1, \sqrt{2}]$ correspond to a capacity release event $(t, i, \alpha_j \frac{c_j}{\beta_j})$ for some $\beta_j \in [1, \sqrt{2}]$ and the sum of the capacities of the release events is $c_i$. We thus obtain

$$\sum_{j : s_{i,j}(t) > 0} \alpha_j c_j = \sum_{(t', i, \alpha_j \frac{c_j}{\beta_j}) \in Q} \beta_j \alpha_j \frac{c_j}{\beta_j} \leq \sqrt{2} c,$$

so that the upload capacity of each node $i$ is also never exceeded by a factor larger than $\sqrt{2}$. □

Before we proceed to formally analyze 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.




**(a)** Augmented solution $\tilde{S}$

**(b)** Feasible solution $S$

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

We now turn to the approximation guarantee of our algorithm. We first need the following lemma that provides a lower bound on the optimal makespan.

**Lemma 10** *Let $I = (N, \mathbf{c})$ be an instance of the minimum time broadcasting problem with $c_0 \geq c_1 \geq c_2 \geq \cdots \geq c_n$ for all $i = 1, \ldots, n$, and let $\tilde{S}$ be a possibly augmented solution with the following properties:*

1. *Every node $i \in N \setminus \{0\}$ receives the file during some interval $[\tilde{A}_i, \tilde{C}_i]$ at constant rate $\tilde{c}_i \geq c_i$;*
2. *$\tilde{A}_i \leq \tilde{A}_j$ for all $i, j \in N \setminus \{0\}$ with $i \leq j$.*

*Then,*

$$t_0 = \min\Big\{t \geq 0 : \text{ there is } i \in N \text{ with } \tilde{C}_i \leq t \text{ and } \sum_{j \in N} \tilde{s}_{i,j}(t) < c_i\Big\} \leq M^*,$$

*where $M^*$ is the optimal makespan for $I$.*

***Proof*** Let us first consider the case $t_0 < \tilde{M}$, where $\tilde{M}$ is the makespan of the augmented solution. 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 in the augmented solution $\tilde{S}$ 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 $\tilde{X}(t_1) \geq X^*(t_1)$. Since $\tilde{x}_{k_0}(t_0) < 1$ and in $\tilde{S}$, node $k_0$ receives the file in some time interval $[\tilde{A}_i, \tilde{C}_i]$ at rate $\tilde{c}_i \geq c_i$, we have $\tilde{x}_{k_0}(t_1) = 0$, and since the nodes start downloading in the order of their indices, we also have $\tilde{x}_i(t_1) = 0$ for all $i \geq k_0$. Every node $i \in N$ with $\tilde{x}_i(t_1) > 0$ receives data with download rate $\tilde{c}_i \geq c_i$ until time $\tilde{C}_i$. Using $\tilde{z}_i(t_1, t_0) = 0$, we obtain

$$\begin{aligned}\tilde{u}_i(t_1, t_0) &\geq \left(\frac{1}{c_{k_0}} - \frac{1 - \tilde{x}_i(t_1)}{\tilde{c}_i}\right) c_i \\ &= \frac{c_i}{c_{k_0}} + \frac{c_i}{\tilde{c}_i}\left(\tilde{x}(t_1) - 1\right) \\ &\geq \frac{c_i}{c_{k_0}} + \tilde{x}_i(t_1) - 1\end{aligned}$$

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

$$\begin{aligned}X^*(t_0) - X^*(t_1) &\leq \sum_{i \in N} \max\left\{0, \frac{c_i}{c_{k_0}} - 1 + x_i^*(t_1)\right\} \\ &= \sum_{\substack{i \in N \\ x_i^*(t_1) > 1 - \frac{c_i}{c_{k_0}}}} \left(\frac{c_i}{c_{k_0}} - 1 + x_i^*(t_1)\right) \\ &\leq \sum_{\substack{i < k_0 \\ x^*(t_1) > 1 - \frac{c_i}{c_{k_0}}}} \left(\frac{c_i}{c_{k_0}} - 1\right) + X^*(t_1),\end{aligned}$$

where we use $c_i/c_{k_0} \leq 1$ for all $i \geq k_0$. We get

$$\begin{aligned}X^*(t_0) - X^*(t_1) &\leq \sum_{i < k_0} (c_i/c_{k_0} - 1) + \tilde{X}(t_1) \\ &\leq \tilde{X}(t_0) - \tilde{X}(t_1),\end{aligned}$$

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 $t_\ell$, 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 $c_0 = \max_{i \in N} c_i$, then* Scale- Fit *is a $2\sqrt{2}$-approximation.*
2. *If $c_0 < \max_{i \in N} c_i$, then uploading the file to another node and applying* Scale- Fit *is a $(1 + 2\sqrt{2})$-approximation.*

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

***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 $\mathcal{O}(n \log n)$. 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 $\mathcal{O}(n)$ of these operations, and if $Q$ is a priority queue, all operations can be executed in time $\mathcal{O}(\log n)$. Therefore the overall running time of the algorithm is $\mathcal{O}(n \log n)$. □

## 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, termed Spread- Exchange, uses a variant of the Scale- Fit algorithm, developed in the last section, as a subroutine

and yields a 5-approximation. The second algorithm, termed SPREAD- MIRROR- CYCLE, has an approximation guarantee of $2 + \frac{2\lceil \log_2 \lceil n/m \rceil \rceil}{m}$, which is super-constant in general, but better than 5 if the number of packets is large compared to the number of nodes.

## 4.1 The SPREAD-EXCHANGE algorithm

The SPREAD- EXCHANGE algorithm works in two phases. In the first phase, called SPREAD, a modification of the SCALE-FIT algorithm is used to provide each node with a single packet. For better reference, we denote the modified version of the algorithm by SCALE- FIT*. In the second phase, called EXCHANGE, the packets are exchanged between the nodes in a pairwise manner until all nodes own all packets. The second phase can only work efficiently, if the packets are distributed in a suitable pattern among the agents.

For ease of exposition, we will first make some simplifying assumptions on the instance and explain the SPREAD-EXCHANGE algorithm for such simple instances. In the end of this section, we will relax these assumptions. First, we require that the server has the highest capacity and that nodes are ordered by their capacities. Second, we require that the capacities are powers of two. Third, we assume that $\sum_{i \in N} \frac{c_i}{c_0} = 2^\ell$ for some integer $\ell$. Intuitively, this condition ensures that the distribution tree is complete and has height $\ell$. Finally, we assume that $m \leq \ell$, that is, there are not more packets than levels in the distribution tree. An instance that satisfies these properties will be called *simple* as summarized in the following definition.

**Definition 12** An instance $I = (N, \mathbf{c}, m)$ is called *simple,* if it satisfies the following properties

1. $c_0 \geq c_1 \geq c_2 \geq \cdots \geq c_n$,
2. $c_i = 2^{r_i}$ with $r_i \in \mathbb{N}$ for all $i \in N$,
3. $\sum_{i=0}^{n} \frac{c_i}{c_0} = 2^\ell$ for some $\ell \in \mathbb{N}$ with $\ell \geq m$.

### 4.1.1 The SCALE-FIT*-algorithm

Our algorithm uses a variant of the SCALE- FIT-Algorithm to send exactly one packet to each node. Compared to the original SCALE- FIT-Algorithm, we introduce two modifications. The modified algorithm still maintains a queue $Q$ of capacity release events but the events now are quadruples

$$(t, i, c, q) \in \mathbb{R}_{\geq 0} \times N \times \mathbb{R}_{\geq 0} \times [m],$$

indicating that at time $t$ node $i$ has an upload capacity of $c$ and will use this capacity to upload packet $q$.

Second, we observe that for a simple instance, where all capacities are powers of two, the SCALE- FIT* algorithm in line 10 can choose $k'$ such that

$$\sum_{j=k}^{k'} c_j = c,$$

that is, no further scaling of the upload or download capacities is needed. This implies in particular that after removing the entry $(t, i, c, q)$ from the capacity release event queue, the set of entries

$$\bigcup_{j=k}^{k'} \left\{ \left( t + \frac{1}{mc_j}, i, c_j, q' \right), \left( t + \frac{1}{mc_j}, j, c_j, q \right) \right\}$$

is added to the queue. The first event $(t + \frac{1}{mc_j}, i, c_j, q')$ means that for the uploading node $i$ the capacity of $c_j$ will be available for further uploads at time $t + \frac{1}{mc_j}$. The packet $q'$ to be send in these uploads will be specified in Sect. 4.1.2. The second event $(t + \frac{1}{mc_j}, j, c_j, q)$ means that node $j$ will have finished the download of the file at time $t + \frac{1}{mc_j}$ and thus its full capacity of $c_j$ will be available for further uploads. Since each node receives a single packet only, node $j$ with $j \in \{k, \ldots, k'\}$ can send only this packet in further uploads. Further note that the download times are shorter by a factor of $1/m$ since only a part of size $1/m$ is distributed by the algorithm.

### 4.1.2 The distribution tree

When introducing new capacity release events in the event queue $Q$, we have to decide which packet is scheduled for upload. This is trivial for non-server nodes who only receive a single packet during the course of the SCALE- FIT*-Algorithm. To explain how the server decides which packet is uploaded, we introduce the notion of a *distribution tree* $T$ which is build incrementally from the events introduced into $Q$. To this end, let $\mathcal{Q}$ denote the set of all capacity release events $(t, i, c, q)$ added to $Q$ during the course of the algorithm. We build a tree structure $T$ with vertex set $V(T) = \mathcal{Q}$ as follows. The root of $T$ is the tree node $(0, 0, c_0, 1)$ corresponding to the event initially placed in $Q$. Whenever a capacity release event $(t, i, c, q)$ is removed from $Q$, the algorithm afterward adds the set of events

$$\bigcup_{j=k}^{k'} \left\{ \left( t + \frac{1}{mc_j}, i, c_j, q' \right), \left( t + \frac{1}{mc_j}, j, c_j, q \right) \right\} \tag{6}$$

to $Q$. In $T$, we add an edge from the tree node $(t, i, c, q)$ to each of the tree nodes $(t + \frac{1}{mc_j}, i, c_j, q')$ with $j = k, \ldots, k'$ and an edge from $(t, i, c, q)$ to each of the tree nodes $(t + \frac{1}{mc_j}, j, c_j, q)$ with $j = k, \ldots, k'$. We call tree nodes of the form $(t + \frac{1}{mc_j}, j, c_j, q)$ *proper tree nodes* and denote the set

**Algorithm 4:** SCALE- FIT*

**Input**: simple instance $I = (N, \mathbf{c}, m)$
**Output**: partial solution where each node $i \in N \setminus \{0\}$ owns one packet

// server has unused upload capacity $c_0$ at time 0
// server will send packet 1
// add this capacity release to event queue
$Q := \{(0, 0, c_0, 1)\}$;
// node with largest capacity not served yet
$k := 1$ ;
**while** $k \leq n$ **do**
  // take next event from queue
  $(t, i, c, q) := \arg\min_{(t',i',c',q') \in Q} t'$ ;
  $Q := Q \setminus \{(t, i, c, q)\}$ ;
  // assign downloaders
  **if** $\sum_{j=k}^{n} c_j < c$ **then**
    $k' := n$;
  **else**
    **choose** $k'$ such that $c = \sum_{j=k}^{k'} c_j$ ;
  **end**
  // sending rate from $i$ to $j$
  $s_{i,j}^{(q)}(\tau) := c_j$ for all $j \in \{k, \ldots, k'\}, \tau \in [t, t + \frac{1}{mc_j})$ ;
  // add new capacity release events to queue
  $Q := Q \cup \bigcup_{j=k}^{k'} \{(t + \frac{1}{mc_j}, j, c_j, k)\}$;
  **if** $i = 0$ **then**
    $Q := Q \cup \bigcup_{j=k}^{k'} \{(t + \frac{1}{mc_j}, 0, c_j, \min\{q+1, m\})\}$ ;
  **else**
    $Q := Q \cup \bigcup_{j=k}^{k'} \{(t + \frac{1}{mc_j}, 0, c_j, q)\}$;
  **end**
  $k := k' + 1$;
**end**

of proper tree nodes by $V^*(T)$. Note that there is a natural bijection between the set of non-server nodes $i \in N \setminus \{0\}$ and the set of proper tree nodes $v \in V^*(T)$. Specifically, for each non-server node $i \in N \setminus \{0\}$, there is a proper tree node $(C_i, i, c_i, q_i)$, where $q_i$ is the packet send to node $i$. Further, there is also a corresponding non-proper tree node $(C_i, p_i^{(q_i)}, c_i, q')$ with $q' \in [m]$, where $p_i^{(q_i)}$ is the unique node that sends packet $q_i$ to node $i$ during the course of the algorithm.

We proceed to provide useful properties of the distribution tree $T$. For a tree node $v$, let level$(v)$ denote the number of edges from $v$ to the root. Further, for a tree node $v = (t, i, c, q)$, we introduce the notations $t(v) = t$, $i(v) = i$, $c(v) = c$ and $q(v) = q$ as a shorthand for the time, node, capacity and packet associated with the corresponding capacity release event. The following lemma establishes that nodes at higher levels have no larger capacities and higher completion times.

**Lemma 13** *Let $v, w \in V(T)$ with level$(v) <$ level$(w)$. Then, $c(v) \geq c(w)$ and $t(v) < t(w)$.*

***Proof*** We show by induction over $\ell$ that for all tree nodes $v$ with level$(v) = \ell$ and all tree nodes $w$ with level$(w) >$ level$(v)$, the claimed properties hold. The only tree node with level$(v) = 0$ is node $(0, 0, c_0, 1)$. Since $t(0, 0, c_0, 1) = 0$ and, by our preprocessing, $c_0 \geq c_i$ for all $i \in N$, the result follows for all tree nodes with level 0.

Assume the statement is true for all tree nodes $v \in V(T)$ with some fixed level level$(v) = \ell$ and all nodes $w \in V(T)$ with level$(w) >$ level$(v)$. Consider a node $v \in V(T)$ with level$(v) = \ell + 1$ and a node $w \in V(T)$ with level$(w) >$ level$(v)$. Let $v'$ and $w'$ be the parent nodes of $v$ and $w$ in $T$. By the induction hypothesis, we have $t(v') < t(w')$, i.e., the capacity release event $v'$ happens before the event $w'$. Since the algorithm considers the nodes in order of non-increasing capacity, this implies $c(v) \geq c(w)$. Since all nodes all served with full capacity, we also have $t(v) = t(v') + \frac{1}{mc_v} < t(w') + \frac{1}{mc_w}$. □

Lemma 13 implies that all capacity release events of a level happen after all capacity release events of a former level. This implies that the tree $T$ is complete in the sense that the capacity of a tree node is equal to the sum of all capacities of its children. Since $\sum_{i=0}^{n} \frac{c_i}{c_0} = 2^\ell$ with $\ell \in \mathbb{N}$ also the last level is complete.

Next, we note that that each capacity release event $(t, i, c_i, q)$ indicates that node $i$ finishes the download of packet $q$ at time $[m]$. Each such capacity event is mirrored by another event of the form $(t, p_i^{(q)}, c_i, q')$ indicating that the upload capacity of the node $p_i^{(q)}$ from which node $i$ received packet $q$ is free again and ready for further uploads. Specifically, $q' = \min\{q + 1, m\}$ if $p_i^{(q)}$ is the server, and $q' = q$ otherwise. As a consequence, half of the capacity of the tree nodes in each level corresponds to available upload capacity of nodes.

We proceed to show that the joint capacity of all nodes that own a particular packet after running the SCALE- FIT*-Algorithm is distributed very regularly.

**Lemma 14** *Let $I = (N, \mathbf{c}, m)$ be a simple instance and 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

$$V_1^* = \{v \in V^*(T) : \text{level}(v) = 1\}$$

be the set of proper tree nodes in level 1 and let

$$\tilde{V}_1 = \{v \in V(T) \setminus V^*(T) : \text{level}(v) = 1\}$$

be the set of non-proper tree nodes in level 1. Since the total capacity of all nodes equals the capacity of the server plus the total capacity of all proper tree nodes, we have

$$\bar{c} = c_0 + \sum_{r \in V_1^*} \sum_{v \in V^*(T(r))} c(v) + \sum_{r \in \tilde{V}_1} \sum_{v \in V^*(T(r))} c(v).$$

The nodes $v \in V^*(T(r))$ with $r \in V_1^*$ are exactly those that receive packet 1, thus,

$$\bar{c} = c_0 + \bar{c}_1 + \sum_{r \in \tilde{V}_1} \sum_{v \in V^*(T(r))} c(v).$$

Since the trees are complete, we have

$$\begin{aligned}\sum_{r \in \tilde{V}_1} \sum_{v \in V^*(T(r))} c(v) &= \sum_{r \in V_1^*} \sum_{v \in V^*(T(r))} c(v) - \sum_{r \in \tilde{V}_1} c(r) \\ &= \sum_{r \in V_1^*} \sum_{v \in V^*(T(r))} c(v) - c_0\end{aligned}$$

We then obtain $\bar{c} = 2\bar{c}_1$ showing the claim for $k = 1$.

To show the result for $1 < k < m$, we apply this argument recursively in the subtrees of level $k$. Let $V_k^*$ denote the set of proper nodes of level $k$ that are served by the server and let $\tilde{V}_k$ denote the set of non-proper tree nodes in level $k$. Then $\bar{c}_k = \sum_{r \in V_k^*} \sum_{vV^*(T(r))} c(v)$. By induction,

$$\begin{aligned}\bar{c} &= c_0 + \sum_{i=1}^{k-1} \bar{c}_i \\ &\quad + \sum_{r \in V_k^*} \sum_{v \in V^*(T(r))} c(v) + \sum_{r \in \tilde{V}_k} \sum_{v \in V^*(T(r))} c(v).\end{aligned}$$

With the same reasoning as before, we obtain the equality $\sum_{r \in \tilde{V}_k} \sum_{v \in V^*(T(r))} c(v) = \sum_{r \in V_k^*} \sum_{v \in V^*(T(r))} c(v) - c_0$, thus,

$$\begin{aligned}\bar{c} &= \sum_{i=1}^{k-1} \frac{\bar{c}}{2^i} + 2 \sum_{r \in V_k^*} \sum_{v \in V^*(T(r))} c(v) \\ &= \left(1 - 2^{-(k-1)}\right)\bar{c} + 2\bar{c}_k,\end{aligned}$$

hence $\bar{c}_k = \bar{c}/2^k$ showing the results for $1 < k < m$.

For $k = m$, we have

$$\begin{aligned}\bar{c} &= c_0 + \sum_{i=1}^{m-1} \bar{c}_i + \bar{c}_m \\ &= c_0 + (1 - 2^{-(m-1)})\bar{c} + \bar{c}_m\end{aligned}$$

and thus

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

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

#### 4.1.3 The EXCHANGE phase

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 round $k$, the total capacity of nodes that own packets $j \in \{k, \ldots, m\}$ is doubled. Lemma 14 then implies that after round $k$, all nodes own packet $k$, and this packet can be ignored in all further rounds. Moreover, 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 $k' > k$ with a straightforward greedy algorithm, see Algorithm 6 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 14, it is straightforward to show that all nodes will get matched this way.

In round $k$, after all nodes are matched that way, every node has a unique packet $j \in \{k, \ldots, m\}$ and sends this packet $j$ at a rate of $c_{\min}$ to its matched node(s).

#### 4.1.4 The SPREAD-EXCHANGE ALGORITHM

A detailed description of the whole procedure SPREAD-EXCHANGE can be found in Algorithm 5. We first show that it yields a 2-approximation algorithm for simple instances.

**Theorem 15** *For the minimum time broadcasting problem with heterogeneous symmetric capacities and a file distributed in $m$ packets,* SPREAD- EXCHANGE *computes a 2-approximation for simple instances.*

***Proof*** Let $I = (N, \mathbf{c}, m)$ be a simple instance and $M^*(I)$ be its optimal makespan. Let $I_{1/m} = (N, \mathbf{c}, 1)$ be the instance with the same set of nodes and capacities where a single file of size $1/m$ has to be send to all nodes. Since in $I$ all $m$ packets of size $1/m$ have to be send to all nodes, we have $M^*(I_{1/m}) \leq M^*(I)$. Consider the SPREAD phase when the

**Algorithm 5:** SPREAD- EXCHANGE algorithm

**Input**: simple instance $I = (N, \mathbf{c}, m)$
**Output**: 2-approximate solution $S$

**run** SCALE- FIT* ; // spread
$t_1$ := makespan of SCALE- FIT*;
**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**
      $s^{(k)}_{i,j}(t) := c_{\min}$ for all $t \in \left[t_1, t_1 + \frac{1}{mc_n}\right)$ ;
      let $k' > k$ be a packet that is owned by $j$ ;
      $s^{(k')}_{j,i}(t) := c_{\min}$ for all $t \in \left[t_1, t_1 + \frac{1}{mc_n}\right)$ ;
    **end**
  **end**
  $t_1 := t_1 + \frac{1}{mc_n}$ ;
**end**

**Algorithm 6:** Greedy assignment

**Input**: sets $A = \{a_1, \ldots, a_r\}$, $B = \{b_1, \ldots, b_s\}$ such that
$\sum_{a\in A} a = \sum_{b\in B} b$,
$a_i = 2^{r_i}$, $b_i = 2^{s_i}$ with $r_i, s_i \in \mathbb{N}$ for all $i \in N$
**Output**: assignment $(a_i, B_i)_{i\in I}$, $(A_j, b_j)_{j\in J}$ such that
$A = \left(\dot\bigcup_{i\in I} a_i\right)\dot\cup\left(\dot\bigcup_{j\in J} A_j\right)$,
$B = \left(\dot\bigcup_{i\in I} B_i\right)\dot\cup\left(\dot\bigcup_{j\in J} b_j\right)$,
$a_i = \sum_{b\in B_i} b$ for all $i \in I$,
$b_j = \sum_{a\in A_j} a$ for all $j \in J$

$M := \emptyset$ ;
**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 $M$;

SPREAD-EXCHANGE algorithm is run for $I$ and let

$$t_0 = \min\Big\{t \geq 0 : \text{ there is } i \in N \text{ with } x_i \geq \frac{1}{m} \text{ and } \sum_{j\in N}\sum_{k\in[m]} s^{(k)}_{i,j}(t) < c_i\Big\}. \tag{7}$$

be the first point in time during the SPREAD phase when available upload capacity remains idle. During the SPREAD phase, no node receives more than one packet, so we may identify the whole SPREAD phase with a solution for instance $I_{1/m}$. Using Lemma 10, we then obtain that $t_0 \leq M^*(I_{1/m})$ implying that $t_0 \leq M^*(I)$. Since in the SPREAD phase no node starts the download of a packet strictly after $t_0$, we have $t_1 - t_0 \leq \frac{1}{mc_n}$ where $t_1$ is the makespan of the SPREAD phase.

After $t_1$, the SPREAD- EXCHANGE algorithm carries out $m-1$ rounds of mutual exchanges of packets, each of which needs $\frac{1}{mc_n}$ time units. As a consequence, the makespan of the solution computed by SPREAD-EXCHANGE is bounded from above by

$$M^*(I) + \frac{1}{mc_n} + \frac{m-1}{mc_n} \leq 2M^*(I),$$

where we used that $M^*(I) \geq \frac{1}{c_n}$ since the optimal solution has to send the whole file to node $n$. □

We proceed to show that a slight modification of the SPREAD- EXCHANGE algorithm yields a 5-approximation for arbitrary, not necessarily simple, instances.

**Theorem 16** *For the minimum time broadcasting problem with heterogenous symmetric capacities and a file distributed in m packets, the following holds:*

1. *If* $c_0 = \max_{i\in N} c_i$, *there is a* 4*-approximation algorithm.*
2. *If* $c_0 < \max_{i\in N} c_i$, *there is a* 5*-approximation algorithm.*

*In both cases, the running time is* $\mathcal{O}(mn \log n)$.

***Proof*** Let $M^*(I)$ denote the optimal makespan of the original instance and let $c_{\max} = \max_{i\in N} c_i$. If $c_0 < c_{\max}$, we spend $1/c_0$ time units to send the whole file from the server node 0 to a node $i^*$ with maximal capacity, remove node 0 from the instance and treat $i^*$ as the new server. Using that $1/c_0 \leq M^*(I)$, it suffices to show 1.

To this end, let the nodes be numbered such that $c_0 \geq c_1 \geq c_2 \geq \cdots \geq c_n$. Consider the instance $\underline{I}$ where all nodes' capacities are rounded down to the next power of two, i.e., $\underline{I} = (N, (\underline{c}_i)_{i\in N}, m)$ where $\underline{c}_i = 2^{\lfloor \log_2 c_i \rfloor}$. By appropriate scaling of the instance, it is without loss of generality to assume that $c_n = \underline{c}_n$. Let $\ell \in \mathbb{N}$ and $n' \in \{0, \ldots, n\}$ be such that

$$\ell = \left\lfloor \log_2 \left( \sum_{i=0}^{n} \frac{\underline{c}_i}{\underline{c}_0} \right) \right\rfloor \quad \text{and} \quad \sum_{i=0}^{n'} \frac{\underline{c}_i}{\underline{c}_0} = 2^\ell.$$

For the subset of nodes $N' = \{0, \ldots, n'\}$, consider the instance with rounded capacities $\underline{I}' = (N', (\underline{c}_i)_{i\in N'}, m)$. Let $p, r \in \mathbb{N}$ be such that $m = p\ell + r$ with $r < \ell$.

The general structure of the algorithm is follows. We conduct $p$ rounds of SPREAD- EXCHANGE for $\underline{I}'$ in each of which we distribute $\ell$ packets to the nodes in $N'$. Then, we conduct a final round of SPREAD- EXCHANGE in which we distribute $r < \ell$ packets, so that after the $p+1$ rounds all $m$ packets are distributed to the nodes in $N'$. Finally, we send the whole file from nodes in $N'$ to nodes in $N \setminus N'$. For the analysis of this algorithm, we distinguish the cases $p = 0$ and $p > 0$.

*First case* $p = 0$. Consider the run of SPREAD-EXCHANGE on the instance $\underline{I}' = (N', (\underline{c}_i)_{i\in N'}, m)$ with a subset of nodes and rounded capacities. As in the proof of Theorem 15, consider the time $t_0$ during the single SPREAD phase defined in (7) when one of the nodes starts to become idle. We claim that $t_0 \le 2M^*(I)$.

To see this, consider again the relaxation $I_{1/m}$ of $I$ where only a single file of size $1/m$ has to be distributed. We have $M^*(I_{1/m}) \le M^*(I)$. We consider the solution for the SPREAD phase for $\underline{I}'$, double all upload and download rates and only send packet 1. By construction, this yields a 2-augmented solution for $I'_{1/m}$. By adding arbitrary (feasible) downloads for the nodes in $N \setminus N'$, we may extend this solution into a solution for $I_{1/m}$. Note that by doubling all upload and download rates, all completion times are halved, but otherwise, no nodes become idle earlier than in the original solution.

By Lemma 10, $t_0 \le 2M^*(I_{1/m})$ and, thus, $t_0 \le 2M^*(I)$. Since in the SPREAD phase no node starts sending a packet strictly after $t_0$, the SPREAD phase ends not later than $M^*(I) + \frac{1}{\underline{c}_{n'}}$. In the EXCHANGE phase, the algorithm performs $(m-1)$ rounds of exchange of length $\frac{1}{m\underline{c}_{n'}}$. Overall, we see that the time needed to run the SPREAD- EXCHANGE algorithm is at most

$$\begin{aligned} 2M^*(I) + \frac{1}{m\underline{c}_{n'}} + \frac{m-1}{m\underline{c}_{n'}} &= 2M^*(I) + \frac{1}{\underline{c}_{n'}} \\ &\le 2M^*(I) + \frac{1}{\underline{c}_n} \\ &= 2M^*(I) + \frac{1}{c_n} \le 3M^*(I). \end{aligned}$$

After the SPREAD- EXCHANGE algorithm, we perform a greedy matching between the nodes $N'$ that received the file and the nodes $N \setminus N'$ that did not receive any portion of the file yet. Since $\sum_{i\in N'} \underline{c}_i \ge \sum_{i\in N\setminus N'} \underline{c}_i$ and all capacities are powers of two, there is a greedy assignment in which all nodes $i \in N \setminus N'$ receive the file at their full (rounded down) capacity $\underline{c}_i$. The assignment can be computed with Algorithm 6. This additional step takes additional

$$\frac{1}{\underline{c}_n} = \frac{1}{c_n} \le M^*(I)$$

time units. We need at most $3M^*(I)$ time units for the SPREAD- EXCHANGE algorithm and $M^*(I)$ time units for sending the file to the nodes in $N \setminus N'$, so that the we obtain a 4-approximation for the case $p = 0$.

*Second case* $p > 0$. For the instance $\underline{I}'$ with a subset of nodes and rounded capacities, the algorithm performs $p > 0$ rounds of SPREAD- EXCHANGE distributing $\ell$ packets each followed by one round distributing $r$ packets. Consider one of the such round during which we distribute $s \in \{\ell, r\}$ packets. In the SPREAD part, the maximum number of subsequent uploads of the server is $\ell$ and each upload takes at most

$$\frac{1}{m\underline{c}_{n'}} \le \frac{1}{m\underline{c}_n} = \frac{1}{mc_n}$$

time units. Further, no download starts strictly after the server has finished its last upload. Thus, the SPREAD phase of this round takes at most $\frac{\ell+1}{mc_n}$ time units. In the EXCHANGE phase, we perform $s-1$ rounds of exchange requiring further $\frac{s-1}{mc_n}$ time units. In total, all $p+1$ rounds of SPREAD- EXCHANGE need

$$\begin{aligned} &(p+1)\frac{\ell+1}{mc_n} + p\frac{\ell-1}{mc_n} + \frac{r-1}{mc_n} \\ &= (p+1)\frac{\ell}{mc_n} + p\frac{\ell}{mc_n} + \frac{r}{mc_n} \\ &= (p+1)\frac{\ell}{mc_n} + \frac{1}{c_n} \\ &\le \frac{3}{c_n} \\ &\le 3M^*(I) \end{aligned}$$

time units.

As in the first case, we spend additional $\frac{1}{c_n} \le M^*(I)$ time units to send the file from the nodes in $N'$ to the nodes in $N \setminus N'$ yielding a 4-approximation.

We now turn to the running time of the algorithm. For the SPREAD phase, we need to run SCALE- FIT* which has a complexity of $\mathcal{O}(n \log n)$. Note that even when SCALE- FIT* is run multiple times for different sets of packets, it actually suffices to run the algorithm only one time and adapt the solutions by changing the packets that are disseminated. For the EXCHANGE phase, we conduct $m-1$ rounds of exchanges for each of which we need to compute an assignment. The greedy algorithm has a complexity of $\mathcal{O}(n \log n)$. Sending the file to the nodes in $N \setminus N'$ requires the solution of a single greedy assignment problem, so that we obtain the overall complexity of $\mathcal{O}(mn \log n)$. □

We note that in a situation where in the beginning the packets are spread all over the nodes in the network, we still obtain a 5-approximation. In the beginning, all packets are sent to a node with largest capacity, which then acts as a server. This sending does not take longer than $M^*(I)$.

We conclude this section with an observation, relating our result to that of Mundinger et al. (2008).

***Remark 17*** 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 Mundinger et al. (2008). (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. The new 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 proceed to 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 $i_{n_1}^1$. Then, if $m > n_1$, nodes $i_1^1, \ldots, i_{m-n_1}^1$ 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 $i_k^j$ sends one of its packets to node $i_{k+1}^j$, except for $i_{n_j}^j$, which sends to $i_1^j$. 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 18** SPREAD- MIRROR- CYCLE *yields a* $(2 + 2\lceil \log_2 \lceil n/m \rceil \rceil / m)$*-approximation in time* $\mathcal{O}(nm)$.

***Proof*** Adding up the time spent on the single phases, the total makespan of the solution computed by SPREAD- MIRROR-CYCLE is bounded by

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

$$\leq 2M^* + \frac{2\lceil \log_2 \lceil n/m \rceil \rceil}{m} M^*,$$

where $M^*$ is the optimal makespan. The partitioning of the set can be done in time $\mathcal{O}(n)$, the sending rates for the SPREAD phase are computed in time $\mathcal{O}(m)$. Then, there are $\mathcal{O}(k) = \mathcal{O}(n/m)$ MIRROR steps, each of which can be computed in time $\mathcal{O}(m)$. Finally, there is the CYCLE phase, in which each node sends $m - 1$ packets. Computing the sending rates of this phase thus takes time $\mathcal{O}(nm)$, 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. Moreover, other objectives such as the weighted sum of completion times seem interesting and deserve further study.

**Funding** Open Access funding enabled and organized by Projekt DEAL.



## References

Arkin, E., Bender, M., Fekete, S., Mitchell, J., & Skutella, M. (2006). The freeze-tag problem: How to wake up a swarm of robots. *Algorithmica, 46*(2), 193–221.

Arkin, E., Bender, M., Ge, D., Hejoseph, S., & Mitchell, S. (2003). Improved approximation algorithms for the freeze-tag problem. In *Proceedings of the 15th annual ACM symposium on parallel algorithms and architectures (SPAA)* (pp. 295–303).

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., et al. (2009). Above the clouds: A Berkeley view of cloud computing. Tech. rep., University of California at Berkeley.

Bar-Noy, A., Guha, S., Naor, J., & Schieber, B. (2000). Message multicasting in heterogeneous networks. *SIAM Journal on Computing, 30*(2), 347–358.

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

Ezovski, G., Tang, A., & Andrew, L. (2009). Minimizing average finish time in P2P networks. In *Proceedings of the 28th IEEE international conference on computer communications (INFOCOM)* (pp. 594–602).

Fan, B., Lui, J. C. S., & Chiu, D. M. (2009). The design trade-offs of bittorrent-like file sharing protocols. *IEEE/ACM Transactions on Networking, 17,* 365–376.

Garey, M., & Johnson, D. (1979). *Computers and intractability*. New York, NY, USA: W. H. Freeman.

Goetzmann, K. S., Harks, T., Klimm, M., & Miller, K. (2011). Optimal file distribution in peer-to-peer networks. In *Proceedings of the 22nd international conference on algorithms and computation (ISAAC)* (pp. 210–219).

Hedetniemi, S. T., Hedetniemi, S. M., & Liestman, A. (1998). A survey of gossiping and broadcasting in communication networks. *Networks, 18,* 129–134.

Khuller, S., & Kim, Y. A. (2007). Broadcasting in heterogeneous networks. *Algorithmica, 48*(1), 1–21.

Könemann, J., Levin, A., & Sinha, A. (2005). Approximating the degree-bounded minimum diameter spanning tree problem. *Algorithmica, 41*(2), 117–129.

Kumar, R., & Ross, K. (2006). Peer assisted file distribution: The minimum distribution time. In *Proc. 1st IEEE workshop on hot topics in web systems and technologies* (pp. 1–11).

Kwon, O. H., & Chwa, K. Y. (1995). Multiple message broadcasting in communication networks. *Networks, 26*(4), 253–261. https://doi.org/10.1002/net.3230260409.

Mehyar, M., Gu, W., Low, S., Effros, M., & Ho, T. (2007). Optimal strategies for efficient peer-to-peer file sharing. In *Proceedings of the IEEE international conference on acoustics, speech and signal processing (ICASSP)* (Vol. 4, pp. 1337–1340).

Middendorf, M. (1993). Minimum broadcast time is NP-complete for 3-regular planar graphs and deadline 2. *Information Processing Letters, 46*(6), 281–287.

Miller, K., & Wolisz, A. (2011). Transport optimization in peer-to-peer networks. In *Proceedings of the 19th international conference on parallel, distributed and network-based processing (PDP)*, (pp. 567–573)

Mundinger, J., Weber, R., & Weiss, G. (2008). Optimal scheduling of peer-to-peer file dissemination. *Journal of Scheduling, 11,* 105–120. https://doi.org/10.1007/s10951-007-0017-9.

Qiu, D., Srikant, R.: Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *Proc. ACM conf. applications, technologies, architectures, and protocols for comput. comm. (SIGCOMM)* (pp. 367–378).

Ravi, R.: Rapid rumor ramification: Approximating the minimum broadcast time. In *Proceedings of the 35th annual IEEE symposium on foundations of computer science* (pp. 202–213).