Broadcasting Algorithms in Peer-to-Peer Data Distribution Systems

by

Seng Keat Teh

SCS Undergraduate Senior Honors Thesis

Faculty Advisors:
Bruce MacDowell Maggs
David Godbe Andersen

School of Computer Science
CARNEGIE MELLON UNIVERSITY

May 2007

Abstract

Our focus is on the study and development of efficient algorithms for the task of broadcasting within peer-to-peer (P2P) data distribution systems. The purpose of broadcasting is to completely distribute a target set of data pieces from a set of initial sources to all peers within the distribution system. Thus, findings from the study of the problem of broadcasting have useful implications for the task of data delivery and file distribution within these systems.

Our work was initially motivated by the desire to develop different data distribution algorithms than those currently employed by popular Internet-based P2P systems. Current P2P file distribution systems such as BitTorrent employ game theoretic motivated algorithms and replication heuristics in the face of distributing files over the Internet to participating peers that are autonomous, self-serving and possibly dishonest. We hope to instead develop efficient data distribution algorithms that would be employed in peer-to-peer networks such as internal or private networks, storage area networks and networks of parallel storage devices where a degree of centralization, coordination and dependability is available.

We believe that the task of broadcasting is a fair theoretical abstraction of this problem. Past work in the field has studied certain basic versions of the broadcasting problem extensively. We will instead focus our efforts on *Disjoint Multi-Source Broadcast* and *Arbitrary Multi-Source Broadcast*, two variants of broadcasting that were previously studied very little.

Contents

List of Figures

Chapter 1

Introduction

Our work in this thesis will attempt to approach the data distribution problem in peer-to-peer systems using techniques from broadcasting and gossiping problems. Current peer-to-peer data distribution systems employed over the Internet utilize game theoretic motivated algorithms to induce cooperation from other autonomous Internet peers that are often self-serving and possibly dishonest. There are however many real-world situations from computer systems that can benefit from the idea of peer-to-peer (P2P) data distribution. Internal or private networks, storage area networks and networks of parallel storage devices, where a degree of centralization, coordination and dependability is available, have frequent need for efficient distribution and replication of data objects.

1.1 Motivation

We were originally motivated to develop algorithms for efficient data distribution within the context of Similarity-Enhanced Transfer (SET), an enhancement developed for peer-to-peer file distribution systems to exploit available similar files as additional sources [15]. SET is also deployed as a core transfer mechanism within Data-Oriented Transfer (DOT), a data transfer service for client applications that separates content negotiation from the task of transferring data [19].

Given that SET now augments the set of available file sources with additional sources from files of exploitable similarity at the level of file chunks, we wanted to know what algorithms would best take advantage of this new resulting scenario and whether we could do better than current algorithms employed by most Internet-based P2P systems.

Most peer-to-peer file distribution algorithms fall under a trichotomy of *peer selection* algorithms, *piece selection* algorithms and *flow control* algorithms [12]. The BitTorrent P2P file distribution system, which has become the *de facto* benchmark

by which all newly developed peer-to-peer data distribution systems are compared to, utilizes *Tit-for-Tat* (TFT) in combination with *Optimistic Unchoking* for both peer selection and flow control while employing a *Local Rarest First* (LRF) heuristic for piece selection [5]. Its peer selection algorithm, *Tit-for-Tat*, is a game theoretic motivated heuristic that addresses the Prisoner's Dilemma-like conditions of most Internet-based peer-to-peer file distribution networks where participating peers are often selfish and self-serving [5, 18]. Its *Local Rarest First* (LRF) piece selection heuristic functions to promote file piece diversity and has been found to work very well in practice in other P2P systems such as Bullet-Prime [12].

While BitTorrent performs well in practice, several weaknesses with the system and its algorithms have been highlighted. It has been found to face a *first blocks problem* that results in a slow startup of the file distribution process [14]. Its TFT algorithm also fails to prevent systematic unfairness in terms of the volume of data served across nodes [4]. Several suggestions for improvements have been made for both BitTorrent's TFT and LRF algorithms [4, 18]. These findings show that while current P2P data distribution algorithms work fairly well in practice, there are still opportunities for continued improvements and the development of better algorithms.

1.2 Approach

In order to better understand the performance of P2P file distribution systems, we examined different theoretical models of such systems that could provide theoretical justifications for the performance and use of current P2P algorithms while shedding light on where they might be improved. We also attempted to initially approach the problem from a different direction, considering the issue of perhaps effective placement of source nodes given a description of demands by peers in a P2P system [1].

We found a wide variety of different approaches towards modeling peer-to-peer file distribution systems [7, 8, 13, 16, 17, 18] through fluid-based models, queueing models and state-based models.

A few efforts attempted to describe peer-to-peer systems using fluid models with an exponential decreasing peer arrival rate [8], fluid replication at downloader peers [13] or through a steady state analysis [16]. A number of efforts also described P2P systems using queueing models. One peer-to-peer queueing model took into account the underlying physical network topology by modeling delays in routers as a single-class open queueing network and peers as processor sharing queues [17]. Another utilized a multiple class closed queueing network that was flexible enough to model different architectures of P2P systems [7]. A third approach was the use of a state-based model where states reflected degrees of file download completion and the behavior of peers were modeled differently according to the state they were

in. Transfer rates to the next succeeding state were modeled as a continuous time Markov chain [18].

We finally decide to take a different approach to the data distribution problem in peer-to-peer systems by abstracting the task as a general *problem of broadcasting* in a peer-to-peer network. The *broadcast problem* is defined as the task of getting a set of messages currently held by one member (a node) of a communication network to every other member of the network [6, 11]. Here, we abstract the set of file pieces as the messages held by a peer.

We find that while peer-to-peer data distribution systems are commonly deployed over the Internet where such systems must deal with self-serving, autonomous peers through the use of game theoretic algorithms to induce cooperation, the idea and benefits of peer-to-peer data distribution is widely applicable to a whole class of situations where a degree of centralization, coordination and cooperation might be available. Internal or private networks, storage area networks and networks of parallel storage devices are examples of such situations that have frequent need for the efficient distribution and replication of data objects.

Furthermore, the *broadcast problem* stands as a very interesting theoretical problem by itself due to its wide applicability. The problem of *broadcasting* and one of its specific variants, *gossiping*, are important in the design of communication protocols in various types of networks for they abstract a large class of communication problems in distributed systems, distributed memory multiprocessor systems and parallel computation problems [2, 3].

Past work has focused on simple variants of the broadcasting problem and more general versions such as *multicasting* [11] and the *Data Migration Problem* [9, 10]. The *Data Migration Problem*, which we will review briefly in the next chapter, is the most general version of the task of data distribution in a peer-to-peer network and has been proven to be NP-hard. In the broadcasting problem instances that we are interested in, we start out with having initial copies of every file piece distributed among some set of peers functioning as the *initial source nodes*. The goal then is to replicate these file pieces to every other node since the goal of peers in a typical P2P file distribution network is to download the entire target file being distributed.

Under the *Data Migration Problem*, each file piece may not necessarily be demanded by all nodes, rather, its *destination nodes*, the nodes that require such a file piece, can be any arbitrary subset of the participating peers. Thus, every instance of the broadcast problem is an instance of the *Data Migration Problem*, though not vice versa.

The focus of our study will be on *Disjoint Multi-source Broadcast* and *Arbi-*

trary Multi-source Broadcast, two variants of the broadcasting problem that to our knowledge have not been studied yet in past work. Our end goal is to develop data distribution algorithms that would minimize the number of timesteps that it takes for both cases to complete their respective file distributions.

We currently have no knowledge of the computational complexity of both problems. While we could utilize the approximation algorithms that have been developed for the *Data Migration Problem* [9, 10] since every variant of the broadcast problem is a data migration problem, we believe that we can develop algorithms with better run times by focusing on the problem at a more specific level.

We will now define the model and definitions used throughout this paper, give a brief review of past work done in the area of broadcasting problems, and then present the focus of our work.

1.3 Model and Definitions

We first define the communication model we assume for peer-to-peer networks discussed in this paper. Our communication model is the *Half-Duplex One-Port 1-Message* model, a common assumption among past work on broadcasting [2, 6, 9, 10, 11].

Two communication models are often used when analyzing about problems in broadcasting, gossiping and multicasting. Under the *Full-Duplex* or telephone model, on a connection between two nodes, both end nodes can function as sender and receiver simultaneously, allowing a bidirectional exchange of file pieces. Under the *Half-Duplex* or mail/telegraph model, exactly only one node can be a sender and one a receiver, permitting only a unidirectional transfer of file pieces.

Under the *One-Port* constraint, no node can be engaged in more than one connection during a single timestep. Under the *1-Message* constraint, each node can only either send or receive a single file piece. We assume that the underlying communication graph is a complete graph, that is any node/peer can connect to any other node/peer.

The *makespan* of a distribution is its completion time, the number of timesteps or rounds required before the file distribution completes and all nodes in the peer-to-peer network have received all file pieces. The *makespan* of an algorithm is the completion time of the distribution under the algorithm.

We outline the following definitions for parameters or variables for any given

broadcast problem.

N = Total number of peers/nodes in the network
K = Number of initial sources at $t = 0$
L = Total number of file pieces composing file to be broadcasted to all peers

Let $\mathbb{K}$ denote the set of initial K source nodes.

S_t^i is the set of nodes that have file piece i at the start of timestep t where $1 \leqslant i \leqslant L$. S_1^i thus denotes the set of initial nodes that hold file piece i, so $S_1^i \subseteq \mathbb{K}$.

P_t^i denotes the *file piece set* of node i at start of timestep t or the set of file pieces held by node i at the start of timestep t. There are thus 2^L different possible *file piece sets* that a node could have, where L is the total number of file pieces that compose the file being distributed. A full source node would thus have a complete *file piece set* while a node that has no file pieces has an empty *file piece set*.

Let $T_{H1,S}(n, k, l)$ represent the makespan or the number of timesteps/rounds that it takes to complete the distribution of a file made up of l file pieces to all n nodes with the initial condition of k initial sources. Subscript S denotes the initial condition that there is exactly only one initial copy of each of the l file pieces. Subscript $H1$ denotes the the *Half-Duplex One-Port 1-Message* assumption.

A *configuration* at timestep t is an assignment of file piece transfer connections that will occur simultaneously during timestep t. A *configuration* assigns which nodes to serve as *senders* or *source nodes* during that timestep, which nodes as *receivers* or *destination nodes* and which file pieces are to be replicated on each file piece transfer connection.

Under the *Half-Duplex One-Port 1-Message*, there can be at most $\lfloor N/2 \rfloor$ simultaneous connections and thus at most $\lfloor N/2 \rfloor$ file pieces can be replicated during any given timestep.

For a given timestep t, when less than $\lfloor N/2 \rfloor$ nodes have at least a single file piece, and a *configuration* assigns each of these nodes to transfer a file piece to a node that has none, we say that the *configuration* is *maximally busy*, as such a *configuration* maximizes the number of file pieces being distributed at timestep t. We say that the the network is *maximally busy* at timestep t if its *configuration* at timestep t is also *maximally busy*. A network cannot be *maximally busy* if its number of nodes with at least one file piece is at least $\lfloor N/2 \rfloor$ but it could be *maximumly busy*.

A *maximumly busy configuration* for timestep t is a set of connections between nodes such that all $\lfloor N/2 \rfloor$ file piece transfer capacities is fully utilized at timestep

t. We say that the the network is *maximumly busy* at timestep t if its *configuration* at timestep t is also *maximumly busy*.

An algorithm that ensures that the network is *maximally busy* when it cannot be *maximumly busy*, and *maximumly busy* when it can, for every timestep preceding the terminal round is thus optimal and results in a data distribution process with the lowest makespan as no algorithm can do any better at each timestep.

Chapter 2

Previous Work

2.1 On the Problem of Gossiping

The *gossip problem* is a variant of the *broadcast problem* in which each node begins with a unique piece of information or file piece that needs to be distributed and thus broadcasted to every other node. Gossiping is in effect an *all-to-all* broadcast, every node is part of the initial K sources, and $K = N$. There is exactly one initial copy and one initial source for each file piece, and each node/peer holds a different file piece. Thus, $L = K = N$.

The problem of *gossiping* is one of the simplest and earliest studied versions of the *broadcast problem*. Using a well-known finding from graph theory about the edge chromatic number for complete graphs [2], the makespan of the gossip problem is

$$T_{H1,S}(N,N,N) = \begin{cases} 2(N-1) & \text{if } N \text{ is even} \\ 2N & \text{if } N \text{ is odd} \end{cases}$$

The simple idea is that each edge color represents a *configuration* for some timestep, and on each connection, it takes two timesteps for both endnodes to exchange file pieces. An extension under $Full - Duplex$ is thus trivial [3],

$$T_{F1,S}(N,N,N) = \begin{cases} N-1 & \text{if } N \text{ is even} \\ N & \text{if } N \text{ is odd} \end{cases}$$

where subscript $F1$ denotes a *Full-Duplex One-Port 1-Message* assumption.

2.2 On the Problem of Broadcasting

2.2.1 Single-Source Scenario

The problem of *Single-Source Broadcast* is a simple variant of the broadcasting problem where one peer serves as the single initial source ($K = 1$) holding an entire copy of the file to be distributed. From this single initial source, the goal is to broadcast all L file pieces that compose the file to all N nodes in the peer-to-peer data distribution network.

Work by Farley [6] on the *Single-Source Broadcast* problem found the distribution to complete within

$$T_{H1,S}(N, 1, L) = \lfloor \log_2 N \rfloor + \left\lceil \frac{L(N-1) - 2^{\lfloor \log_2 N \rfloor} + 1}{\lfloor N/2 \rfloor} \right\rceil$$

This is a lower bound on the makespan for the *Single-Source Broadcast* situation, since we know that for the first $\lfloor \log_2 N \rfloor$ timesteps, $2^{\lfloor \log_2 N \rfloor} - 1$ is the maximum amount of file pieces that can be transmitted within this period. At every round after this stage but preceding the terminal round, all capacities of the network is fully utilized. During the terminal round, the distribution is completed. Hence, any other distribution algorithm cannot complete any faster.

When N is odd, the makespan reduces to become

$$T_{H1,S}(N, 1, L) = \lfloor \log_2 N \rfloor + 2L - 1 \qquad \text{when } N \text{ is odd}$$

2.2.2 L-Multi-Source Scenario

Under the *L-Multi-Source Broadcast* case, the goal is to distribute a file made out of L pieces to all N nodes in the network but with the initial starting conditions that each initial copy of the L file pieces is held at a different node at the start. Hence, $K = L$, we have L initial sources, each holding a single different file piece of the L file pieces. Each file piece then has exactly one initial copy located at a different initial source and no initial source can hold more than one file piece at the start.

Khuller et al. [11] developed a polynomial-time algorithm that completes *L-Multi-Source Broadcast* within

$$T_{H1,S}(N, L, L) = \left\lceil \log_2 \frac{N}{L} \right\rceil + 2L$$

A high-level description of the data distribution algorithm is that it is essentially made up of two phases, where during the first phase, the algorithm partitions the N

nodes into L groups of equal size. Each of the initial L sources will be a member of a different group. Without loss of generality, assume that the initial source for piece i is a member of group i where $1 \leqslant i \leqslant L$. Each of these initial sources will then broadcast their file piece to every other node in its group. In the second phase, N/L groups of size L are formed by picking one node from each of the L groups from the first phase. Each of these groups is essentially an instance of the gossiping problem, since every group member holds a different file piece. We complete the distribution by solving these gossip instances in parallel [11].

A special case of *L-Multi-Source Broadcast* is $K = L = N$, the distribution then becomes a problem of *Gossiping*.

2.3 On the Problem of Multicasting

A more challenging extension to the problem of broadcasting is the problem of *multicasting*. Instead of broadcasting the entire file and thus all L file pieces to all N nodes, under *multicasting*, each file piece is demanded by some subset of the N nodes, and these subsets may overlap each other, meaning that a node can request for some of the file pieces and not necessarily all L pieces. We denote D_i to represent the subset of N nodes that demand file piece i where $1 \leqslant i \leqslant L$.

2.3.1 Single-Source Multicast

Single-Source Multicast is a more challenging extension of the *Single-Source Broadcast* problem. Instead of broadcasting all L file pieces, we now have a set of demanders, D_i, for each file piece i to which we must replicate and send the file piece to. Again, we start from the same initial conditions as in *Single-Source Broadcast*, we have $K = 1$ and a single initial source node that holds all L file pieces. Thus, the number of initial sources for a piece i, $|S_1^i| = 1$.

A polynomial time algorithm that is a 2-approximation has been developed for *Single-Source Multicast* [11]. The algorithm finishes in at most $OPT + L$ rounds where OPT is the minimum number of rounds required for the problem. The total makespan of the algorithm is at most

$$\leq \max_{1 \leqslant i \leqslant L} (i + \lfloor \log |D_i| \rfloor) + L$$

2.3.2 L-Multi-Source Multicast

L-Multi-Source Multicast, similar to its counterpart version in broadcasting, *L-Multi-Source Broadcast*, begins with the initial conditions of having L initial sources,

each holding one of the L file pieces that compose the file. The goal is to distribute a file piece i to nodes that are member of D_i, the set of peers that demand file piece i.

Finding a schedule that minimizes the number of rounds of a given instance of the *L-Multi-Source Multicast* problem is NP-hard. Khuller et al. [11] showed a polynomial-time reduction from a restricted version of 3SAT known to be NP-hard into an instance of the *L-Multi-Source Multicast* problem.

A polynomial time $(3 + o(1))$-approximation algorithm was developed [11] with a makespan of

$$\max_{1 \leqslant i \leqslant L} (log|D_i|) + 2\beta + O(\sqrt{\beta})$$

where

$$\beta = \max_{j=1,\cdots,N} |\{i|j \in D_i\}|$$

β thus represents the largest number of file pieces currently demanded for this instance of the *L-Multi-Source Multicast* problem. β is also an upper bound on the number of different sets D_i to which a peer j can belong to.

Both $\max_{1 \leqslant i \leqslant L}(log|D_i|)$ and β are lower bounds on the *L-Multi-Source Multicast* problem. Hence, the algorithm is a $(3 + o(1))$-approximation.

2.3.3 The General Data Migration Problem

The most general abstraction for the task of data distribution in peer-to-peer networks is the *Data Migration Problem*. Let $S_i = S_0^i$ denote the set of initial sources for file piece i where $1 \leqslant i \leqslant L$. The *Data Migration Problem* is then stated as follows, for every file piece i, we need to utilize the initial sources for this file piece, S_i, to replicate file piece i to peers that are member of D_i, the set of nodes that demand piece i.

Hence, the initial source nodes for a file piece i and its destinations nodes are both arbitrary subsets of all N nodes. There can thus be more than one initial copy of piece i, a node be an initial source for more than one file piece and a node may demand some subset of the L file pieces. Thus, every variant of the broadcasting, multicasting and gossiping problems are instances of the *Data Migration Problem*.

Khuller et al. [10] have shown that the *Data Migration Problem* is NP-hard through a polynomial-time reduction from the problem of edge coloring with the smallest number of colors, which we know to be NP-hard, to an instance of the problem. An improved $(6.5 + o(1))$-approximation algorithm has been developed for the problem [9].

Chapter 3

Our Focus

Our main focus is investigating and possibly developing efficient algorithms and heuristics for two specific variants of the broadcasting problem: *Disjoint Multi-source Broadcast* and *Arbitrary Multi-source Broadcast*. We assume nodes, representing peers in a network, follow the *Half-Duplex One-Port 1-Message* model, symbolically denoted as $H1$. Under this assumption, a node may only sustain a connection to exactly one other node at a timestep (the *One-Port or Pairwise* constraint), one endnode serves exclusively as a sender while the other is a receiver (the *Half-Duplex* constraint) on any given connection, and the sender node may only transmit a single file piece (the *1-Message* constraint) to the receiver on its connection.

On a single timestep, the network may possibly have multiple matchings that are disjoint from each other in that no two matchings share any endnodes or vertex. Each matching represents a valid connection between two nodes, and thus these matchings represent connections that can occur simultaneously within a given timestep.

3.1 Validity of Model Assumptions

The broadcast problem is a fair abstraction of the peer-to-peer data distribution problem, providing a reasonable approximation to the scheduling challenges faced by peer-to-peer data distribution. In truth, real world instances of peer-to-peer data distribution must contend with other technical concerns, such as the underlying data transfer protocol used which is most commonly TCP/IP, differing bandwidth rates on outgoing connections (uplink bandwidth) and incoming connections (downlink bandwidth) between any two peers and the effects of the underlying physical topology of the network. The abstraction of a complete graph where a node is connected to every other node is a reasonable approximation since peer to peer networks are

usually sustained as an application-level overlay network over the Internet where a connection between any two nodes is made possible through TCP/IP.

While it is true that in real peer-to-peer data distribution networks, a peer may be connected to a number of other peers simultaneously, the *One-Port or Pairwise* constraint reflects the desire to avoid node-congestion. It is quite reasonable to assume that on a very minute discrete time level, a peer is exchanging data pieces with one other node at a time.

The *Half-Duplex* constraint may perhaps be a weaker approximation to real world peer-to-peer systems than a *Full-Duplex* assumption. Under the *Half-Duplex* assumption, on a given connection, the direction of data transfer is unidirectional: only one endnode may act as a sender while the other must be a receiver. With *Full-Duplex*, data transfer becomes bidirectional: endnodes can function as both sender and receiver. Under a *1-Message* constraint then, the connection can sustain a maximum transfer of two data pieces. For example, the peer-to-peer file distribution system BitTorrent reflects a *Full-Duplex* situation, since one of its core algorithms, Tit-for-Tat, gets a peer to reciprocate any useful data piece it receives from another peer with a data piece that this other peer desires. However, again on a very fine grained time interval level, the exchange of data pieces between two peers can be seen as a half-duplex or unidirectional situation.

Finally, since peer-to-peer data distribution systems commonly split files to be distributed into numerous equivalent pieces of the same size, the constraint of only being able to transfer a single file piece or *1-Message* over a connection seems fairly reasonable. As an example, BitTorrent divides a file being distributed into equal-sized *pieces* of 256 KB and these *pieces* are further subdivided into *blocks* of 16 KB sizes [4]. BitTorrent downloads *pieces* concurrently from multiple peers, but obtains the *blocks* of a *piece* from a specific peer. Thus, at the most basic level, it is reasonable to approximate such behavior as a *1-Message* limitation.

The *Half-Duplex One-Port 1-Message* model would at best fit an approximation to real world peer-to-peer system on a very fine grained time scale level.

3.2 Investigating New Variants of the Broadcasting Problem

In the last section, we explored previous work on the broadcasting problem within the *Half-Duplex One-Port 1-Message* context, where efforts have been primarily focused on the simple *Single-Source Broadcasting* problem where $K = 1$, the *Gossiping* problem where $L = K = N$ and *L-Multi-Source Broadcasting* where $L = K$. How-

ever, little work has been done in the field to address situations where the number of initial sources is different from 1 or L, and where the distribution of the initial first copies of the L file pieces varies among the K initial sources.

We will focus on new variants of the broadcasting problem, specifically what we term as *Disjoint Multi-source Broadcast* and *Arbitrary Multi-source Broadcast*.

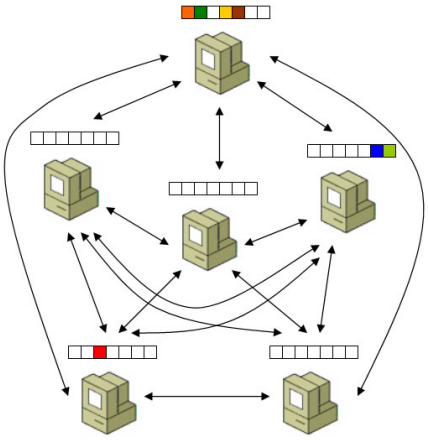

Figure 3.1: Disjoint Multi-source Broadcast with $N = 6$, $K = 3$, $L = 7$

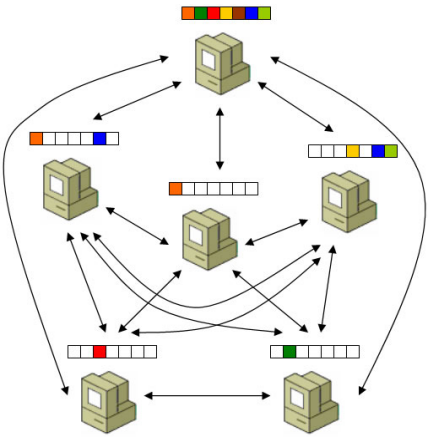

Figure 3.2: Arbitrary Multi-source Broadcast with $N = 6$, $K = 6$, $L = 7$

Disjoint Multi-source Broadcast is a variant of the broadcasting problem where the state of the peer-to-peer data distribution network starts off with K initial sources, each of which holds a disjoint subset of the L file pieces that make up the entire file to be distributed. Thus, no two initial sources may hold the same file piece, the subset of file pieces that each initial source node holds is disjoint from each other and there is exactly a single copy of each of the L file pieces at the start of the distribution process. When $K = 1$, this reduces to the *Single-Source Broadcasting* situation while when $K = L$, this becomes the *L-Multi-Source Broadcasting* problem studied extensively in previous work. Our focus under *Disjoint Multi-source Broadcast* is then to study situations where $1 < K < L$.

Arbitrary Multi-source Broadcast is a variant of broadcasting where $1 < K \leqslant N$ and each of the L file pieces will have at least a single copy but can also have multiple copies initially among the K initial sources. Hence, the set of initial nodes that have file piece i at the start of the data distribution process, S_1^i at $t = 1$, is some arbitrary subset of the K initial sources.

Little has been explored about both variants in current literature and work on broadcasting problems.

3.3 Abstractions of Real World Situations

We choose to study both the *Disjoint Multi-source Broadcast* and the *Arbitrary Multi-source Broadcast* cases as these are abstractions of specific real world situations faced by peer-to-peer data distribution systems.

As we discussed earlier, the peer-to-peer data distribution problem can be reasonably abstracted as the broadcast problem and modeled by a complete graph in which nodes represent the participating peers. Most if not all current peer-to-peer file distribution systems employ *segmented donwloading* in which different pieces of the file are downloaded simultaneously from different peers. A file being distributed in the system is thus represented and modeled as a collection of file pieces that must all be obtained by a node to complete its download of the file. In certain literature, file pieces are termed as *commodities* to be distributed, and the problem thus becomes a *commodity distribution* problem.

In real world peer-to-peer distribution scenarios, N, the model parameter that denotes the number of peers demanding the target file being distributed, is usually in the hundreds or thousands. K, the parameter denoting the number of initial starting sources is usually in the tens or hundreds. L, denoting the number of file pieces that make up the entire file, is usually largest in value among all the model parameter, reflecting the reality that files being distributed using peer-to-peer data distribution are often very large files in the sizes of hundreds of megabytes or several gigabytes. The size of a file piece is usually in the magnitude of kilobytes; BitTorrent for example divides files into 256 KB-sized *pieces* that are further subdivided into 16 KB-sized *blocks* [4]. Thus, L is usually a value in range of thousands and above.

3.4 Disjoint Multi-source Broadcast

The *Disjoint Multi-source Broadcast* case approximates a real world peer-to-peer distribution network scenario with a starting state where none of the initial K source nodes holds the entire copy of the file being distributed but all file pieces are available collectively from these initial sources. Thus, the file download can still proceed to completion.

The goal of *Disjoint Multi-source Broadcast* is to distribute all L file pieces to all N nodes in the network. The distribution network starts with K initial sources where $1 < K < L$. Each of these K initial sources hold a disjoint subset of these L file pieces.

Let $T_{H1,S}(N, K, L)$ where $1 < K < L$ denote the makespan of the *Disjoint Multi-*

source Broadcast situation under the *Half-Duplex One-Port 1-Message* model.

3.4.1 A Few Lower Bounds

We first derive a few clear lower bounds on the completion time of *Disjoint Multi-source Broadcast* case.

Lemma 3.4.1. *For a peer-to-peer distribution network with N nodes and K initial sources distributing a file made out of L file pieces, the makespan of the distribution must take at least the same number of rounds that it takes to get every node to obtain its first file piece.*

$$T_{H1,S}(N, K, L) \geq \left\lceil \log_2 \frac{N}{K} \right\rceil$$

Proof. Every node that currently holds at least one file piece should replicate one of its pieces to a node that has none. By doing so at every timestep, we double the number of nodes with at least a single file piece. We know we cannot do any better than this in increasing the number of nodes with a file piece. The peer-to-peer distribution network begins with K initial sources and the number of nodes with at least one file piece can at most double at every next timestep. Hence, this takes exactly $\lceil \log_2 N/K \rceil$ before all N nodes gets a file piece. □

Lemma 3.4.2. *Let P_t^i be the set of file pieces currently held by node i at timestep t. Let $i \in \mathbb{K}$, where $\mathbb{K}$ is the set of the initial K source nodes. Then, $|P_0^i|$ denotes the size of the disjoint subset of the L file pieces that is held by initial source i at $t = 0$, the start of the data distribution process. Then, the makespan must be at least as large as the maximum number of initial file pieces held among the K initial sources at the start of the process.*

$$T_{H1,S}(N, K, L) \geq \max_{i \in \mathbb{K}} |P_0^i|$$

Proof. Since on every timestep under the *Half-Duplex One-Port 1-Message* assumption, a node can send out only a single file piece, it takes at least $|P_0^i|$ timesteps to distribute all the initial $|P_0^i|$ different file pieces held by an initial source node source i where $i \in \mathbb{K}$. Since every node needs to download the entire file and hence needs to acquire all L file pieces, the distribution process must take at least as long as it takes to introduce all the different file pieces from the largest subset of the L file pieces held among the K initial sources at the start of the process. □

Lemma 3.4.3. *The makespan of Disjoint Multi-source Broadcast under the Half-Duplex One-Port 1-Message assumption with L file pieces, N nodes and K initial must be at least*

$$T_{H1,S}(N,K,L) \geq \left\lceil \frac{L(N-1)}{\lfloor N/2 \rfloor} \right\rceil$$

Proof. Since there are N nodes and each desire the entire file, a total of NL file pieces should be transmitted in this file distribution process. Since we initially began with a single copy of the original file but with these L pieces distributed arbitrarily among the initial K sources, we thus have left another $L(N-1)$ file pieces that needs to be transmitted using the peer-to-peer distribution network.

Within a single timestep, there can be at most $\lfloor N/2 \rfloor$ simultaneous connections and hence at most $\lfloor N/2 \rfloor$ file pieces can be transmitted during a given timestep. Hence, the total number of timesteps for the distribution must take at least the above lower bound. □

We now derive a much better lower bound for the makespan of the *Disjoint Multi-source Broadcast* problem.

Lemma 3.4.4. *The number of timesteps to complete the Disjoint Multi-source Broadcast distribution must be at least*

$$T_{H1,S}(N,K,L) \geq \left\lfloor \log_2 \frac{N}{K} \right\rfloor + \left\lceil \frac{L(N-1) + K - K2^{\lfloor \log_2 N/K \rfloor}}{\lfloor N/2 \rfloor} \right\rceil$$

Proof. We first examine the maximum number of file pieces that can be transmitted within the first $\tau = \lfloor \log_2 N/K \rfloor$ timesteps. The total number of file pieces transmitted in the first τ timesteps can be at most

$$\begin{aligned} &\leq K + 2K + 4K + \ldots + 2^{\tau-1}K \\ &\leq K(1 + 2 + 4 + \ldots + 2^{\tau-1}) \\ &\leq \frac{(2^{\tau}-1)K}{2-1} \end{aligned}$$

since the initial state of the network begins with K initial sources, each holding a subset of the L file pieces that make up the entire file. Hence, during the first timestep, at most K file pieces can be transmitted. In the next time period, we know we can at most double the number of file pieces transmitted if in the previous timestep, we transmitted each of the K file pieces to a different node that had no file pieces, thus resulting in $2K$ nodes in the current time period that can participate in file piece transmissions. We know we can't do better than doubling the number of file pieces sent at every timestep.

Let $\eta = (2^\tau - 1)K = (2^{\lfloor \log_2 N/K \rfloor} - 1)K$. We know that the the network needs to distribute NL file pieces to get the file to all N nodes. However, we initially started with a single copy of each of the L file pieces, albeit distributed disjointly among the initial K sources. Then the number of remaining messages at the start of $t = \tau + 1$ must be at least

$$\geq NL - \eta - L$$

We know that at a given timestep, at most $\lfloor N/2 \rfloor$ simultaneous connections are possible and hence at most $\lceil N/2 \rceil$ file pieces can be distributed at a time. Then, the number of remaining timesteps before all L commodities is distributed to all nodes is at least

$$\begin{aligned}
&\geq \left\lceil \frac{NL - \eta - L}{\lfloor N/2 \rfloor} \right\rceil \\
&\geq \left\lceil \frac{NL - (2^{\lfloor \log_2 N/K \rfloor} - 1)K - L}{\lfloor N/2 \rfloor} \right\rceil \\
&\geq \left\lceil \frac{NL - 2^{\lfloor \log_2 N/K \rfloor} + K - L}{\lfloor N/2 \rfloor} \right\rceil \\
&\geq \left\lceil \frac{L(N-1) + K - 2^{\lfloor \log_2 N/K \rfloor}}{\lfloor N/2 \rfloor} \right\rceil
\end{aligned}$$

Hence, the number of timesteps to complete the *Disjoint Multi-source Broadcast* distribution is at least

$$\begin{aligned}
T_{H1,S}(N,K,L) &\geq \tau + \left\lceil \frac{L(N-1) + K - 2^{\lfloor \log_2 N/K \rfloor}}{\lfloor N/2 \rfloor} \right\rceil \\
&\geq \left\lfloor \log_2 \frac{N}{K} \right\rfloor + \left\lceil \frac{L(N-1) + K - 2^{\lfloor \log_2 N/K \rfloor}}{\lfloor N/2 \rfloor} \right\rceil
\end{aligned}$$

□

If $K = 1$, this reduces to the lower bound for the *Single-Source Broadcast* case.

3.4.2 A Condition for Maximum Busyness

We present a lemma relating file piece frequency and the existence of a *maximumly busy* configuration for a certain state of a peer-to-peer data distribution network at a given timestep. A *maximumly busy* configuration is an assignment of nodes that uses all $\lfloor N/2 \rfloor$ file piece transfer capacities.

We earlier defined a *file piece set* of a node i at timestep t, P_t^n, as the set of the file pieces that node n has currently received so far by timestep t. There are thus 2^L

different possible *file piece sets* that a node could have, where L is the total number of file pieces that make up the file being distributed. P_t^n can be easily described as a L-length binary string where a value of 1 at the i-th bit correspond to node n having the i-file piece by timestep t. A full source node would thus have a complete *file piece set* while a node that has yet to receive any file pieces has an empty *file piece set*.

A connection between two nodes that have the same *file piece set* is idle, since no needed file piece can be provided by one node to the other. But a connection between two nodes with differing *file piece sets* can engage in useful work. Let P_t^A and P_t^B respectively denote the *file piece sets* of nodes A and B at the start of time period t where $P_t^A \neq P_t^B$. There are three possible cases for the difference in file piece sets:

- $P_t^A \subset P_t^B$, thus node B has at least one file piece that node A does not have. Node B will thus be the sender and node A as the receiver on useful connection between both nodes.
- $P_t^B \subset P_t^A$, which is a reverse from before, hence, node A will be the sender while node B becomes the receiver node.
- P_t^A and P_t^B are neither subsets of each other but they may possibly overlap. Hence, both node A and B each possess at least one file piece that the other does not have. Both can be either a sender or receiver though not simultaneously as both due to the *Half-Duplex One-Port 1-Message* assumption.

Let X_t^j denote the number of nodes that has a certain *file piece set* j at timestep t, where $1 < j < 2^L$.

Lemma 3.4.5. *Let file piece set j be the most common set of file pieces held among N nodes at the start of timestep t. Then, if $X_t^j \leq \lceil N/2 \rceil$, then there exists a maximumly busy configuration of connections between nodes such that all $\lfloor N/2 \rfloor$ file piece transfer capacities is fully utilized at timestep t.*

Proof. The proof for this lemma can be broken down into three cases:

Case 1: $X_t^j > \lceil N/2 \rceil$
When N is even, there are more nodes with the *file piece set* j than there are other nodes of different *file piece sets*. When N is odd, there will be insufficient nodes of different *file piece sets* for nodes with *file piece set* j to connect to using all $\lfloor N/2 \rfloor$ file piece transfer capacities.

Case 2: $X_t^j = \lceil N/2 \rceil$
The second case is obvious, there are equal number of nodes that have *file piece set* j and nodes that do not. A pairing between one node from each group will always

be a useful connection, and there are enough such connections that can be formed to utilize all $\lfloor N/2 \rfloor$ file piece transfer capacities.

Case 3: $X_t^j < \lceil N/2 \rceil$
We know that there are X_t^j nodes that have *file piece set* j, hence there are $N - X_t^j$ other nodes that are available to be paired with these X_t^j nodes in useful connections. Thus, a total of $2X_t^j$ nodes are involved in these connections, we then have a remainder of $N - 2X_t^j$ nodes that may or may not be able to form useful connections among themselves. These nodes are not of *file piece set* j but they could all be the same of some other *file piece set*.

We know we have a total of $\lfloor N/2 \rfloor$ file piece transfer capacities available, with X_t^j of these currently utilized. We will show that there will be at least $\lfloor \frac{N}{2} \rfloor - X_t^j$ connections among the X_t^j connections between nodes of *file piece set* j and nodes of other *file piece sets*, such that these nodes of other *file piece sets* will not be holding the same *file piece set* as the remaining $N - 2X_t^j$ nodes.

Assume, without loss of generality, that the remaining $N - 2X_t^j$ nodes are of the same *file piece set* and let this be some *file piece set* k. Certainly, if some of these remaining $N - 2X_t^j$ nodes were to be of different *file piece sets* besides k, then useful connections/pairings can be formed between these nodes and those of *file piece set* k, leaving an even smaller number of nodes that could possibly be idle. Hence, in the worst case scenario, all the remaining $N - 2X_t^j$ nodes are of the same *file piece set* k.

Let D_t^k be the number of nodes that have *file piece set* k from the X_t^j other nodes that were connected to the X_t^j nodes of *file piece set* j.

We know that *file piece set* j is the most common *file piece set* by our assumption. Hence,

$$\begin{aligned}(N - 2X_t^j) + D_t^k &\leq X_t^j \\ D_t^k &\leq X_t^j - (N - 2X_t^j)\end{aligned}$$

Hence, there can be at most $X_t^j - (N - 2X_t^j)$ nodes of *file piece set* k from the X_t^j other nodes connected to those of *file piece set* j. This leave at least

$$X_t^j - (X_t^j - (N - 2X_t^j)) = N - 2X_t^j$$

other nodes that are not of *file piece set* k.

These $N - 2X_t^j$ other nodes are themselves in connections with nodes of *file piece set* k. We merely need to break $\left\lfloor \frac{N-2X_t^j}{2} \right\rfloor = \lfloor \frac{N}{2} \rfloor - X_t^j$ of these current connections

so that we will have $\lfloor \frac{N}{2} \rfloor - X_t^j$ nodes of *file piece set* j and another $\lfloor \frac{N}{2} \rfloor - X_t^j$ nodes that are not of *file piece set* k. These can then be used to form $2\lfloor \frac{N}{2} \rfloor - 2X_t^j$ useful connections with the $N - 2X_t^j$ nodes of *file piece set* k.

The total number of connections now formed is

$$\begin{aligned} &= X_t^j - (\left\lfloor \frac{N}{2} \right\rfloor - X_t^j) + (2\left\lfloor \frac{N}{2} \right\rfloor - 2X_t^j) \\ &= 2X_t^j - \left\lfloor \frac{N}{2} \right\rfloor + 2\left\lfloor \frac{N}{2} \right\rfloor - 2X_t^j \\ &= \left\lfloor \frac{N}{2} \right\rfloor \end{aligned}$$

Thus, we have proven that we can create a *maximumly busy* configuration that utilizes all $\lfloor \frac{N}{2} \rfloor$ file piece transfer capacities at timestep t when $X_t^j < \lceil N/2 \rceil$. □

3.4.3 A General Algorithm and A Set of Heuristics

We now present a general algorithm and a set of heuristics for completing *Disjoint Multi-source Broadcast*. The heuristics outlined will prove helpful in trying as best as possible to achieve *maximumly busyness*, or when not all $\lfloor N/2 \rfloor$ file piece transfer capacities could be fully utilized, *maximal busyness*, during every timestep except the terminating round.

The key idea towards maintaining *maximumly busyness* at every time step is to pack as much useful work as possible that can be done at a given timestep. But for a given timestep and state of the network, it is possible that there will be numerous *maximumly busy* configuration to choose from. One must be chosen such that the configuration leads to a state of the network in the next timestep where it is still possible to construct a *maximumly busy* configuration, unless such a timestep is the terminating round of the distribution.

A General Four Phase Algorithm for Disjoint Multi-source Broadcast

Phase 1. The purpose of the first phase is to ramp up and maximize the network utilization of all available file piece transfer capacities.

For any given network of N nodes, there can be at most $\lfloor N/2 \rfloor$ simultaneous connections and thus file pieces that can be distributed at any one time. At the start, there are K initial sources and at most K simultaneous useful connections that can be constructed, connecting one of these sources with one of the other $(N - K)$ nodes

that hold no file pieces. During *Phase 1*, the network will be *maximally busy* but is unable to be *maximumly busy* while the number of nodes with at least one file piece is less than $\lfloor N/2 \rfloor$ as this implies that the most frequent *file piece set*, the empty set, is held by more than $\lceil N/2 \rceil$ nodes. By Lemma 3.4.5, it will then not be possible to construct a *maximumly busy* configuration.

The goal of *Phase 1* is thus to get at least $\lfloor N/2 \rfloor$ peers or more in the network to have at least one file piece in the least amount of time. We know we cannot do better than by doubling the number of nodes with a file piece at every timestep. To do so, the network must be *maximally busy*, that is every node with a file piece needs to transmit a file piece to some other node that has none.

Phase 1 thus completes within $\left\lfloor \log_2 \frac{N}{K} \right\rfloor$ timesteps.

1. At every timestep, every node with at least one file piece must replicate one of its file pieces to another peer that has no file pieces.

2. Each of the initial K sources hold some subset of the L file pieces that make up the original file. At every timestep, each of these initial sources must replicate a different file piece from their respective subset.

3. When a source reaches its last different file piece, then it should keep on replicating this file piece until the end of *Phase 1*.

For example, let P_0^i where $1 \leqslant i \leqslant K$ denote the disjoint subset of L file pieces held by initial source i. In the first $|P_0^i|$ timesteps, source i must give out a different file piece. Should source i run out of new file pieces to give out before *Phase 1* completes, that is if $|P_0^i| \leq \lfloor \log_2 N/K \rfloor$, then it should keep on distributing the last file piece it had given out previously.

At the end of *Phase 1*, exactly $K2^{\lfloor \log_2 N/K \rfloor}$ peers now have at least a single file piece and

$$\begin{aligned} K2^{\lfloor \log_2 N/K \rfloor} &> K2^{(\log_2 N/K)-1} \\ &> \frac{K2^{\log_2 N/K}}{2} \\ &> \frac{K}{2}\frac{N}{K} = \frac{N}{2} \end{aligned}$$

By the end of *Phase 1*, it is then possible for the network to be *maximumly busy* at the next time step.

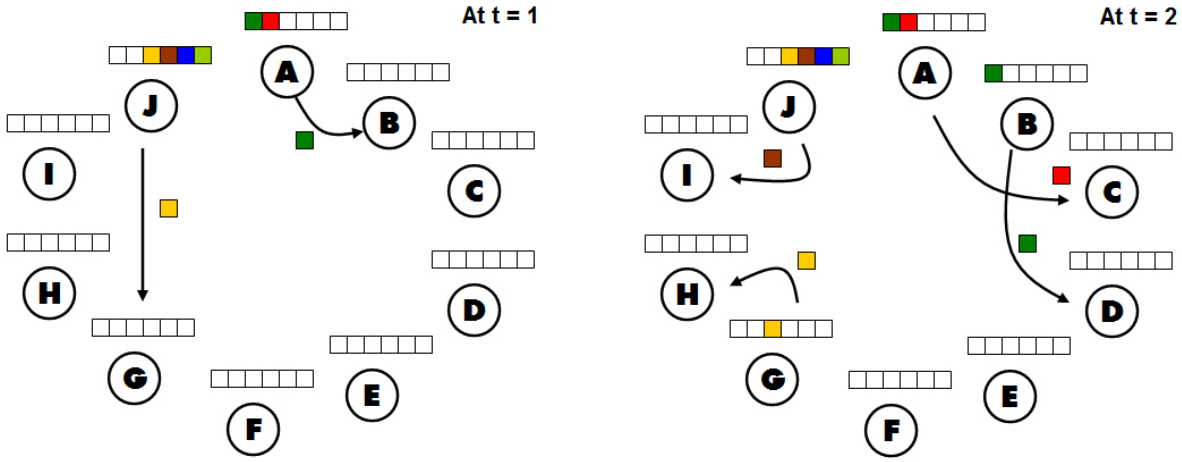

Figure 3.3: Phase 1 for *DMSB* with $N = 10$, $K = 2$, $L = 6$

Phase 2. The second phase focuses on trying to replicate exactly $\lceil N/2 \rceil$ copies of each of the L file pieces while keeping each timestep during this phase *maximumly busy*. Each file piece will have at least $\lceil N/2 \rceil$ copies by the end of *Phase 2*.

Initially, at the end of *Phase 1*, no file piece can be at more than $\lceil N/2 \rceil$ nodes. We know this because *Phase 1* takes $\lfloor \log_2 N/K \rfloor$ timesteps, hence no file piece can be replicated to more than $2^{\lfloor \log_2 N/K \rfloor}$ nodes, which is less than or equal to $2^{\log_2 N/K} = N/K$ nodes with $1 < K < L$.

We utilize two data structures for this phase: a *Frequency Table* that records the current number of copies of each file piece and a *Completion Table* that keeps track of file pieces with more than $\lceil N/2 \rceil$ copies during *Phase 2*.

Phase 2 also aims to maximize the utilization of all $\lfloor N/2 \rfloor$ file piece transfer capacities at each timestep during this phase. Hence, while we try to avoid having more than $\lceil N/2 \rceil$ copies for each of the L file pieces, there are times when this constraint has to be exceeded in order to ensure all $\lfloor N/2 \rfloor$ file piece transfer capacities is utilized. It is these file pieces that will be tracked by the *Completion Table*.

1. Construct the *Frequency Table* and record the number of copies of each of the L file pieces at the start of *Phase 2*.

2. We now need to construct a *maximumly busy* configuration for this current timestep that utilizes all available $\lfloor N/2 \rfloor$ connections. We have $\lfloor N/2 \rfloor$ replication slots to distribute among the L file pieces, and we then identify sources for pieces we select to replicate and their destination nodes.

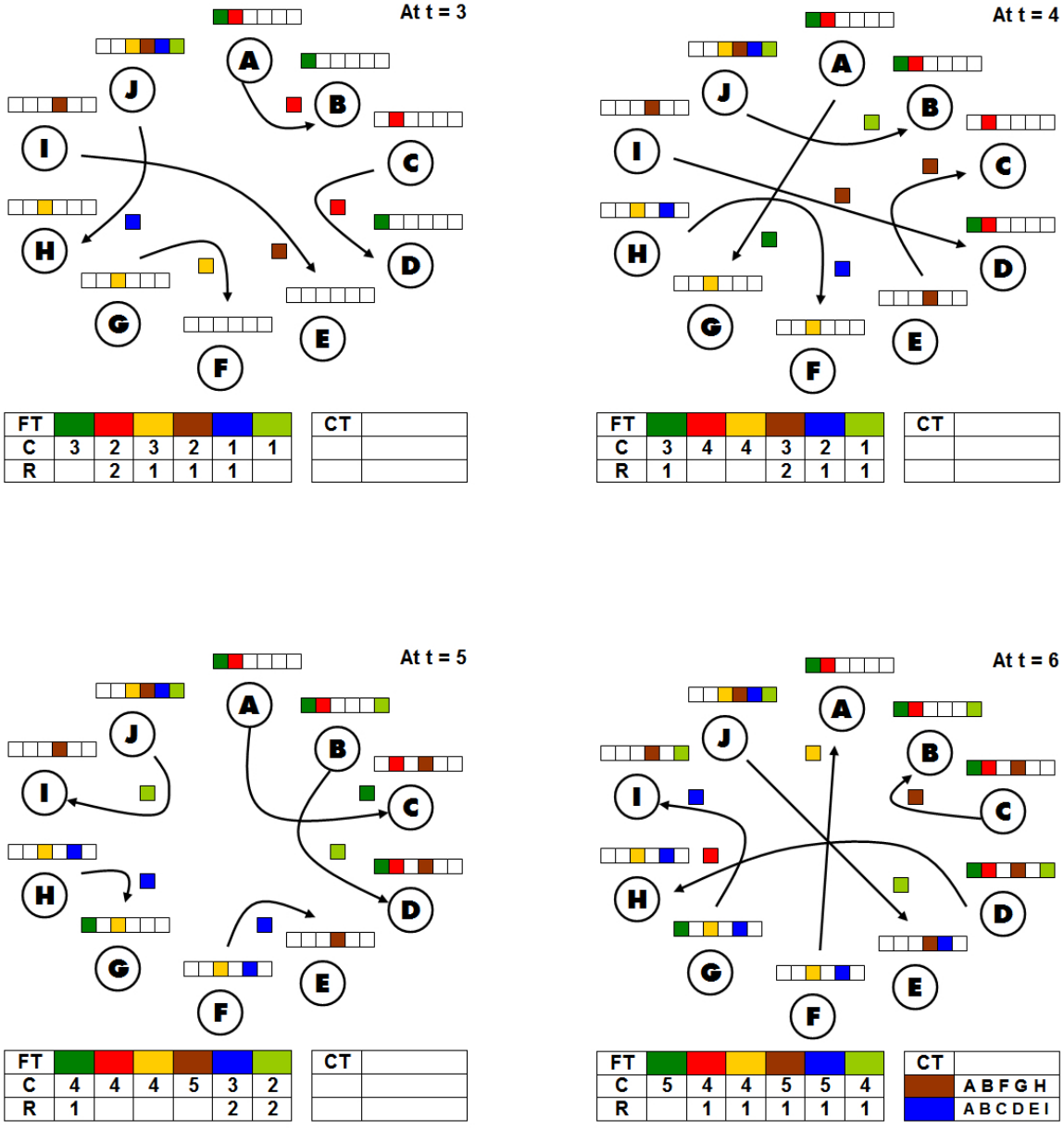

Figure 3.4: Phase 2 for *DMSB* with $N = 10$, $K = 2$, $L = 6$

3. Sort the *Frequency Table* by frequency count. We will only replicate file pieces that currently have less than $\lceil N/2 \rceil$ copies. We are going to select $\lfloor N/2 \rfloor$ pieces to replicate starting with the least frequent file pieces.

4. We thus give priority towards first **replicating the current rarest file pieces** in the distribution network. We start with file pieces with the lowest frequency count, then move on to the next lowest frequency count if we still have available replication slots. We will only replicate file pieces that currently have less than $\lceil N/2 \rceil$ copies.

5. For all file pieces with the same frequency count, we will be fair in our selection and distribute the available replication slots in a *round robin* manner, assigning each of these file pieces one slot at a time and repeating this procedure until we have either

   - Completely allocated all available $\lfloor N/2 \rfloor$ replication slots
   - No more source nodes are available for any of these file pieces with this frequency count
   - For file pieces that still have available source nodes, these file pieces have received enough replication slots such that in the next timestep, they will have $\lceil N/2 \rceil$ copies

6. We then move on to the next lowest frequency count if we still have available replication slots and repeat Step 5 until we are done assigning $\lfloor N/2 \rfloor$ replication slots or find that the next lowest frequency count is at least $\lceil N/2 \rceil$.

7. We will now decide on the source and destination nodes for file pieces that we have selected to replicate. Currently, we have not developed an algorithm that can efficiently produce a selection of source and destination nodes that would fulfill the demands and constraints of each file piece being replicated. We will however outline a few heuristics that we have found to work well in determining such selections in several examples we have tested.

   Let $P_t^{REPLICATE}$ denote the set of file pieces selected so far to be replicated during timestep t. Let R_t^i denote the number of copies of file piece i that will be replicated during timestep t and thus, $0 \leqslant R_t^i \leqslant \lfloor N/2 \rfloor$. Let $S_t^{i,REPLICATE}$ be the set of source nodes selected to replicate file piece i at timestep t, where $i \in P_t^{REPLICATE}$ and $S_t^{i,REPLICATE} \subseteq S_t^i$. Let $D_t^{i,REPLICATE}$ be the set of destination nodes to receive file piece i at timestep t.

   Then any selection of sources and destinations for file piece i where $i \in P_t^{REPLICATE}$ must adhere to the following constraints:

   - $S_t^{i,REPLICATE} \subseteq S_t^i$

- $|S_t^{i,REPLICATE}| = |D_t^{i,REPLICATE}| = R_t^i$
- $D_t^{i,REPLICATE} \cap S_t^i = \emptyset$
- $S_t^{i,REPLICATE} \cap S_t^{j,REPLICATE} = \emptyset$ where $i \neq j$ and $j \in P_t^{REPLICATE}$
- $S_t^{i,REPLICATE} \cap D_t^{j,REPLICATE} = \emptyset$ where $i \neq j$ and $j \in P_t^{REPLICATE}$
- $D_t^{i,REPLICATE} \cap D_t^{j,REPLICATE} = \emptyset$ where $i \neq j$ and $j \in P_t^{REPLICATE}$

While we have not developed a deterministic algorithm that can efficiently produce a selection adhering to the above constraints, we will now outline a few heuristics that we have found to work well in several examples we have examined.

- For a file piece i where $i \in P_t^{REPLICATE}$, if all of its sources, S_t^i have been selected as sources for file piece replication, then any of the $N - |S_t^i|$ other nodes can be selected as destination nodes for file piece i. We label such file pieces i as *sources-fully-engaged* file pieces.
- Then, for file pieces j where $j \in P_t^{REPLICATE}$, if not all of its sources, S_t^j, are selected as sources for file piece replication at this time step, we label these as *sources-partially-engaged* file pieces.
- The heuristic for destination node selection for some piece a where $a \in P_t^{REPLICATE}$ is to select destination nodes from S_t^b, where piece b is some *sources-partially-engaged* file piece. The rationale here is that this does not hurt but may help piece b since this does not decrease piece b's own respective set of possible destination nodes. We must however ensure that there is still enough remaining nodes from S_t^b to be members of $S_t^{i,REPLICATE}$.
- In reverse to how we did replication slots distribution in Step 3 and 4, in destination selection, we give **priority to the most frequent file pieces first**, since logically, these file pieces have a smaller set of possible destination nodes. We employ the heuristic just mentioned earlier, trying at best to select a destination node that is a member of S_t^b where piece b is some *sources-partially-engaged* file piece.

8. By the end of Step 7, we may have remaining replication slots that we were unable to assign if we find that at the start of Step 7, all file pieces with available source nodes already have or will have $\lceil N/2 \rceil$ copies by the next timestep.

9. Let β be the number of remaining replication slots by the end of Step 7. If $\beta = 0$, then skip to Step 12.

10. If $\beta > 0$, we must now allocate these remaining slots. We aim to maximize the utilization of all $\lfloor N/2 \rfloor$ file piece transfer capacities at each timestep during

Phase 2. By the end of Step 7, we have selected $2(\lfloor N/2 \rfloor - \beta)$ nodes as senders and receiver nodes, and have at least 2β nodes left. From these remaining nodes, we need to select β nodes as senders and another β receivers and determine which file pieces will be replicated on these β connections. It is these file pieces on these β connections that will be tracked by the *Completion Table*.

11. Once these file pieces have been determined, insert a new entry for each of these file pieces that are not currently recorded in the *Completion Table*. Every entry in the table will keep track of some file piece h that has exceeded $\lceil N/2 \rceil$ copies during some timestep within *Phase 2* and the set of nodes that have not yet receive piece h.

12. Having at best configured $\lfloor N/2 \rfloor$ file piece transfer connections, we allow these file piece transfers to occur.

13. We now update the *Frequency Table* for file pieces that were replicated during this round. We also update the *Completion Table* if some piece h tracked by this table was replicated as well. In particular, nodes that have received piece h during this timestep if removed from the *Completion Table's* entry for piece h.

14. We repeat the entire algorithm again starting from Step 2 until every of the L file pieces is replicated in at least $\lceil N/2 \rceil$ nodes. We must try at best to ensure that every timestep during *Phase 2* is *maximumly busy* and utilizes all $\lfloor N/2 \rfloor$ file piece transfer capacities.

By the end of *Phase 2* we know that each file piece is replicated in at least $\lceil N/2 \rceil$ nodes. File pieces that have exactly $\lceil N/2 \rceil$ copies will be completely distributed to all N nodes during *Phase 3*. In *Phase 4*, we will then complete the distribution of the remaining nodes with more than $\lceil N/2 \rceil$ copies by the end of *Phase 2*. We will use the information from the *Completion Table* to complete *Phase 4* in the minimal amount of timesteps possible.

Phase 3. The third phase completes the distribution of every file piece i that had exactly $\lceil N/2 \rceil$ copies at the end of *Phase 2*. Let γ where $\gamma \leqslant L$ be the number of such file pieces. Then, *Phase 3* takes exactly γ rounds to complete.

Each of these file pieces currently reside at exactly $\lceil N/2 \rceil$ nodes, a remaining $\lfloor N/2 \rfloor$ nodes desire the file piece and at most $\lfloor N/2 \rfloor$ file piece transfers can occur at a time. Each file piece then takes exactly one timestep to complete its distribution to all N nodes. Each of these γ timesteps are also *maximumly busy* since all $\lfloor N/2 \rfloor$ file piece transfer capacities are utilized. Hence, *Phase 3* is on a whole *maximumly busy*.

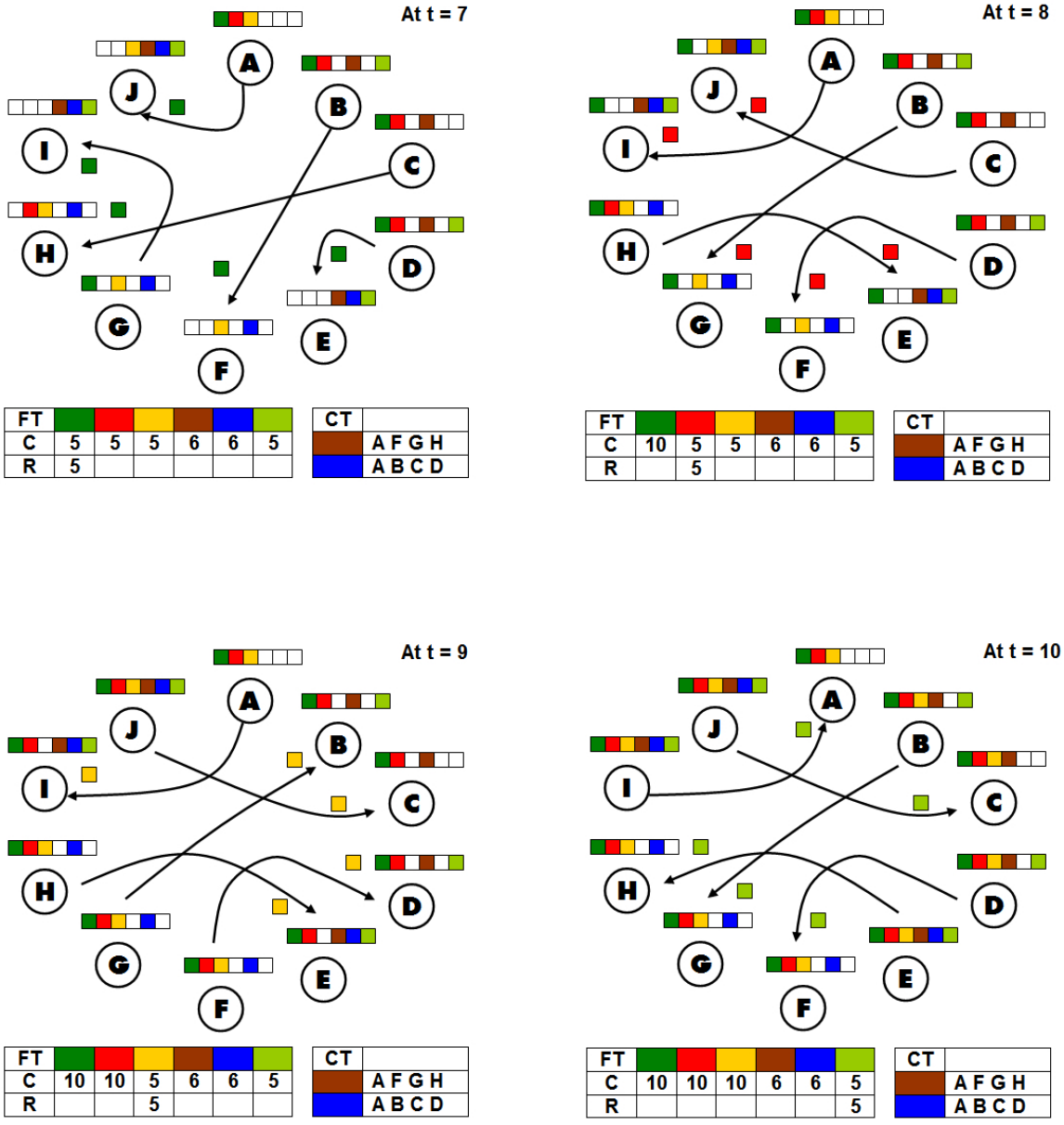

Figure 3.5: Phase 3 for *DMSB* with $N = 10$, $K = 2$, $L = 6$

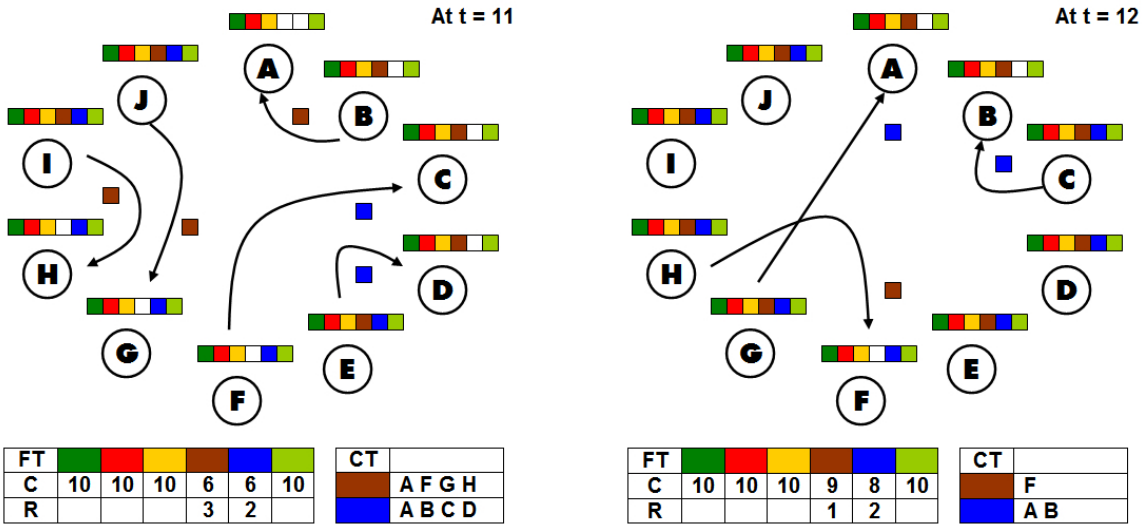

Figure 3.6: Phase 4 for *DMSB* with $N = 10$, $K = 2$, $L = 6$

Phase 4. In the final phase, we complete the distribution of file pieces that exceeded $\lceil N/2 \rceil$ copies by the end of *Phase 2*. The *Completion Table* informs us, for each piece h of these file pieces, which of the N nodes that have yet to acquire piece h. Using this information, we try our best in constructing a *maximumly busy* configuration that utilizes all $\lfloor N/2 \rfloor$ file piece transfer capacities.

We again face the same challenge as we did at Step 7 of *Phase 2* in producing a selection of sender and receiver nodes and the file pieces to replicate that will result in a *maximumly busy* configuration. However, any configuration that is produced must again adhere to the constraints on the sources and destination nodes for each file piece i where $i \in P_t^{REPLICATE}$, as outlined in Step 7 of *Phase 2*.

We again utilize the same heuristics employed in Step 7 of *Phase 2* and the information provided by the *Completion Table* in coming up with a *maximumly busy* configuration for each timestep under *Phase 4*.

1. We give priority towards **replicating the current rarest file pieces** in the distribution network. Rarest pieces are indicated in the *Completion Table* as entries with the largest number of nodes that has yet to receive that file piece.

2. For all file pieces with the same rarity, we will again be fair in our selection and distribute the available $\lfloor N/2 \rfloor$ replication slots in a *round robin* manner until we

    - Completely allocated all available $\lfloor N/2 \rfloor$ replication slots

- No more source nodes are available for any of these file pieces with this frequency count

3. We then move on to the next lowest frequency count if we still have available replication slots and repeat Step 2 until we are done assigning all $\lfloor N/2 \rfloor$ replication slots.

4. With the $\lfloor N/2 \rfloor$ file pieces to replicate identified, we now proceed with destination selection, and again as in Step 7 of *Phase 2*, give **priority to the most frequent file pieces first** that are being replicated during this timestep.

5. We employ the same heuristic of selecting destination nodes for a file piece a where $a \in P_t^{REPLICATE}$ from S_t^b where piece b is a *sources-partially-engaged* file piece. We must again however ensure that there will still be enough remaining nodes from S_t^b to be members of $S_t^{i,REPLICATE}$.

6. Having at best configured $\lfloor N/2 \rfloor$ file piece transfer connections, we allow these file piece transfers to occur.

7. We update both the *Frequency Table* and the *Completion Table*.

8. We repeat the entire algorithm for the next timestep starting from Step 1 until all N nodes have obtained all L file pieces and thus the entire file.

We earlier mentioned that if all preceding timesteps to the terminal round is *maximumly busy* or *maximally busy* when it is not possible to be *maximumly busy*, then *Disjoint Multi-source Broadcast* completes in the number of timesteps outlined by the lowerbound from Lemma 3.4.4. The terminal round itself need not necessarily be *maximumly busy*. Thus, in *Phase 4* we try at best to get all timesteps preceding the terminal round to be *maximumly busy*.

Let γ be the number of file pieces that had exactly $\lceil N/2 \rceil$ copies at the end of *Phase 2*. Then, $(L - \gamma)$ is an upperbound on the completion time of *Phase 4*, since there are $(L - \gamma)$ file pieces that had more than $\lceil N/2 \rceil$ copies at the end of *Phase 2*, and each of these pieces clearly need only one timestep to complete its distribution to all N during *Phase 4*.

Thus, L is an upperbound on the combined total completion time of *Phase 3* and *Phase 4*.

3.4.4 Interesting Findings

Our study of the *Disjoint Multi-source Broadcast* problem has resulted in the identification of a lower bound on its makespan, the development of a general algorithm and a set of heuristics that have worked well in practice in solving this particular

variant of the broadcasting problem. We have also obtained a few interesting findings relating to the properties of the *Disjoint Multi-source Broadcast* problem.

Finding 1: *The degree of asymmetry of the distribution of the initial copies of the L file pieces among the K initial sources does not affect the lowerbound on the makespan of Disjoint Multi-Source Broadcast.*

First, we find that the asymmetry of the distribution of the initial L file pieces over the K initial sources have no bearing on the lower bound of the makespan of the distribution. From Lemma 3.4.4, we had derived that

$$T_{H1,S}(N,K,L) \geq \left\lfloor \log_2 \frac{N}{K} \right\rfloor + \left\lceil \frac{L(N-1)+K-K2^{\lfloor \log_2 N/K \rfloor}}{\lfloor N/2 \rfloor} \right\rceil$$

An algorithm for *Disjoint Multi-source Broadcast* that is *maximally busy* during the first $\lfloor \log_2 N/K \rfloor$ timesteps and *maximumly busy* for all succeeding timesteps prior to the terminal round would be able to achieve the lower bound on the makespan from Lemma 3.4.4.

We see that the lower bound we derived in Lemma 3.4.4 is only affected by three parameters: N, the total number of nodes, K, the number of initial sources and L the number of file pieces that make up the file being distributed. Thus, two instances of *Disjoint Multi-source Broadcast* with the same values for parameters N, K and L but differing in how the initial copies of the L file pieces are distributed among the K initial sources will have the same lower bound on the makespan.

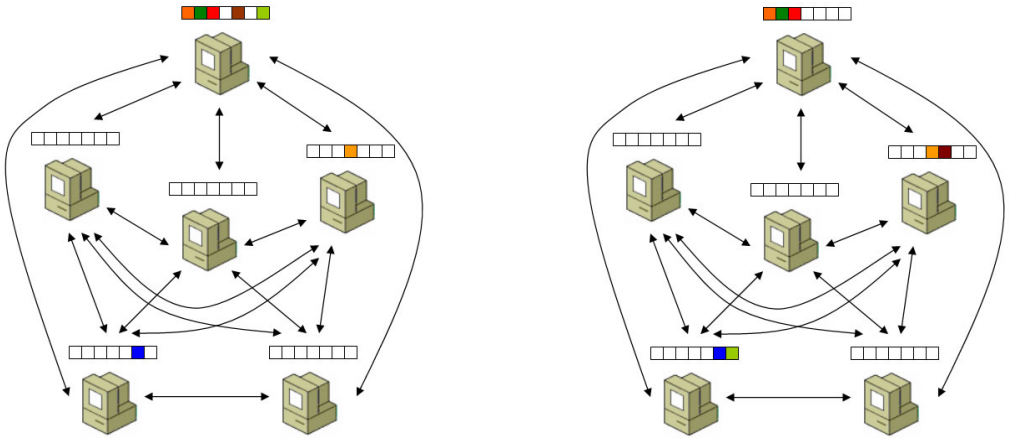

Figure 3.7: Both initial Disjoint Broadcast setups result in equal makespans

We present two examples where $N = 6$, $K = 3$ and $L = 7$ but differ in the initial distribution of the L file pieces among the K initial sources. The first example has greater asymmetry, where the majority of the file pieces are concentrated at a single node. The second example has a uniform distribution of the initial file pieces. Both examples have equal makespans.

We see that in these particular examples, there is no difference in the time to complete between the two *Disjoint Multi-source Broadcast* instances where they only differ in how the L pieces were distributed over the K initial sources. Intuitively, it would seem better if the L pieces were more uniformly distributed among the initial K sources and that this would help facilitate a faster completion time for the distribution. This is simply not the case with the provided examples.

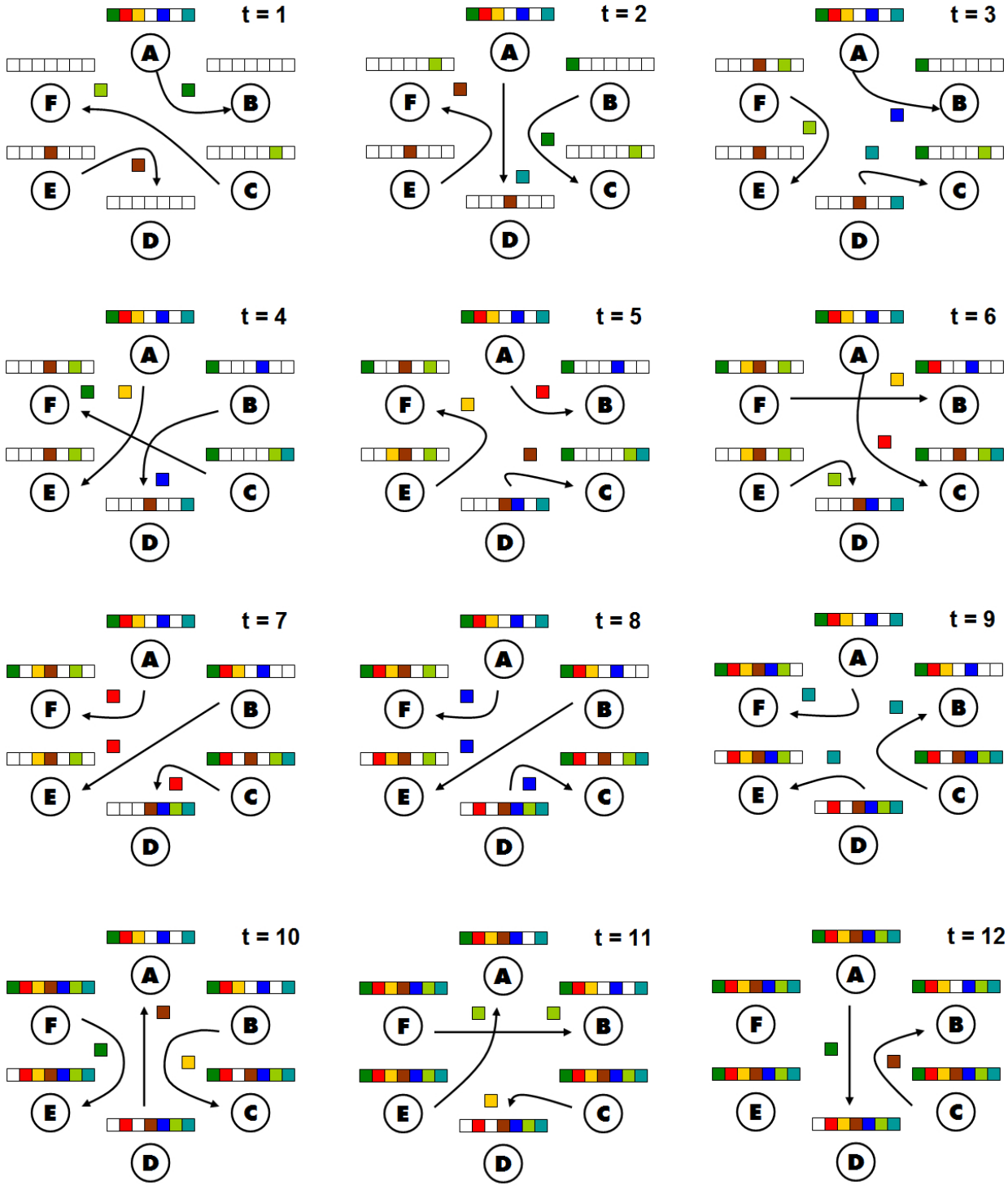

Figure 3.8: Example of $N = 6$, $K = 3$ and $L = 7$ with greater asymmetry in the initial distribution of file pieces among initial sources

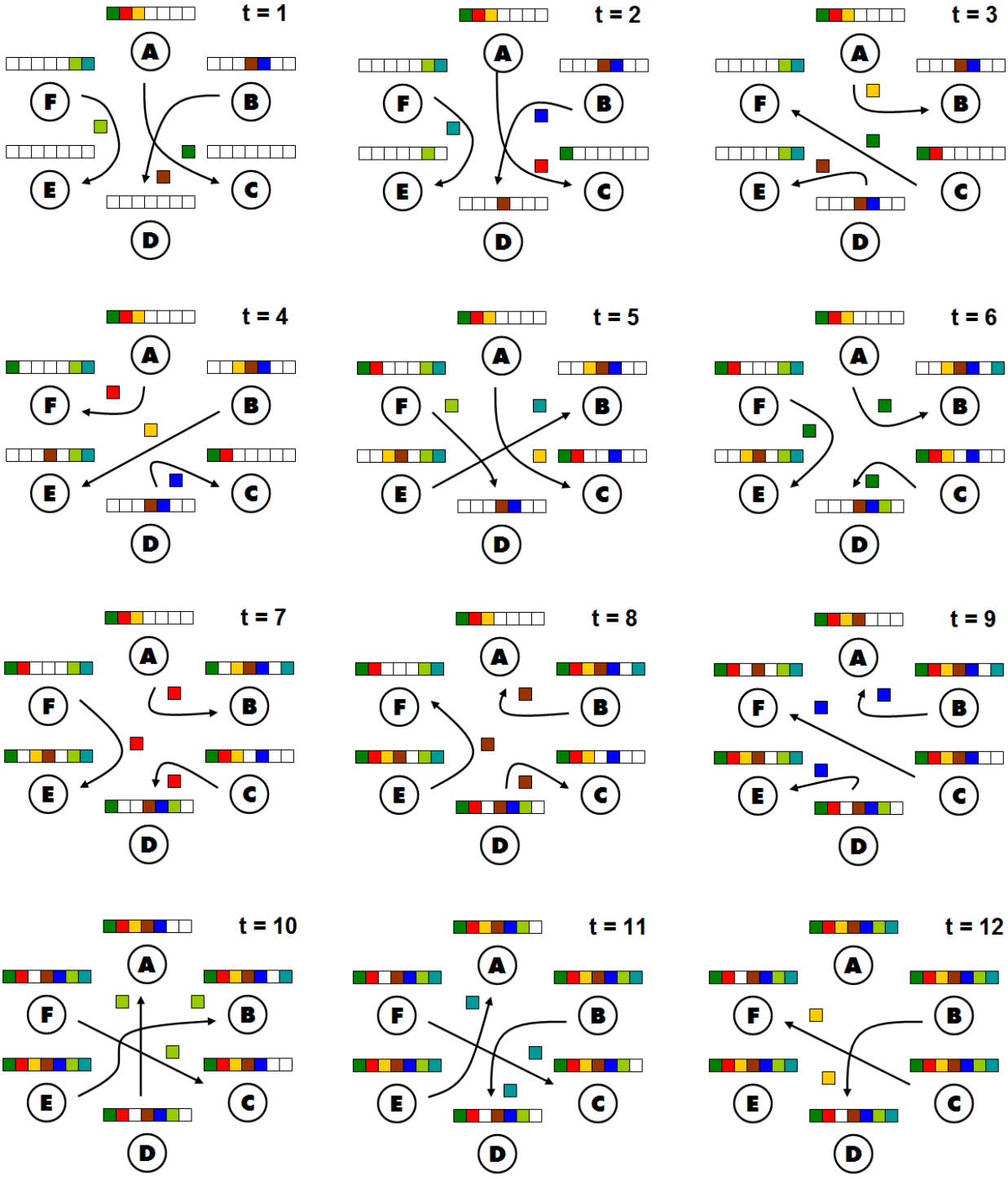

Figure 3.9: Example of $N = 6$, $K = 3$ and $L = 7$ with uniform distribution of initial copies of file pieces among initial sources

Finding 2: *A local greedy maximizing busyness algorithm does not necessarily lead to a global optimum for the Disjoint Multi-Source Broadcast Case.*

Employing a local greedy algorithm that tries to ensure each local node is engaged in any available useful file piece transfer during the current timestep is insufficient towards creating a global optimum. Simply, having every node locally be engaged in some useful local work may still result in an overall non-optimal solution.

An optimal solution is one that minimizes the makespan of *Disjoint Multi-source Broadcast*. If an algorithm is able to ensure that each timestep prior to the terminal round is *maximumly busy* (utilizing all $\lfloor N/2 \rfloor$ file piece transfer capacities) or *maximally busy* when it is not possible to be *maximumly busy* (if there are less than $\lfloor N/2 \rfloor$ nodes with at least one file piece), such an algorithm is optimal as no other algorithm can do any better during each of these timesteps.

This thus motivates the intuition and idea that employing a greedy local maximizing busyness algorithm may provide an optimal solution to the problem. We have however found counterexamples where in the first few timesteps, the peer-to-peer data distribution network is *maximumly busy*, only to result in a later non-terminal timestep where the network can no longer do so. This implies that some thought must go into the configuration of nodes to ensure that the succeeding state of the network can still be *maximumly busy*.

Figure 3.10 is one counterexample we have found where the first few rounds are *maximumly busy*, but the distribution process progresses to a later non-terminal timestep at $t = 7$ where the distribution effectively reduces into a *Single-Source Broadcasting* problem. The given example completes in 16 timesteps while its optimal solution completes in 15 timesteps.

We have seen that being *maximumly busy* so far may still lead to a non-optimal solution in the long run. In the provided example, what led to the distribution process degrading into a *Single-Source Broadcasting* problem at $t = 7$ were the choices of file pieces that were replicated during the first few timesteps from $t = 1$ to at $t = 6$.

A common set of file pieces, pieces A, B, C and D were continuously replicated during these timesteps, eventually resulting in all nodes obtaining all of such pieces. However, *Node 3* continued to hold rare pieces that had yet been obtained by other nodes. Hence, the distribution effectively degraded into a *Single-Source Broadcasting* instance with *Node 3* functioning as the single-source.

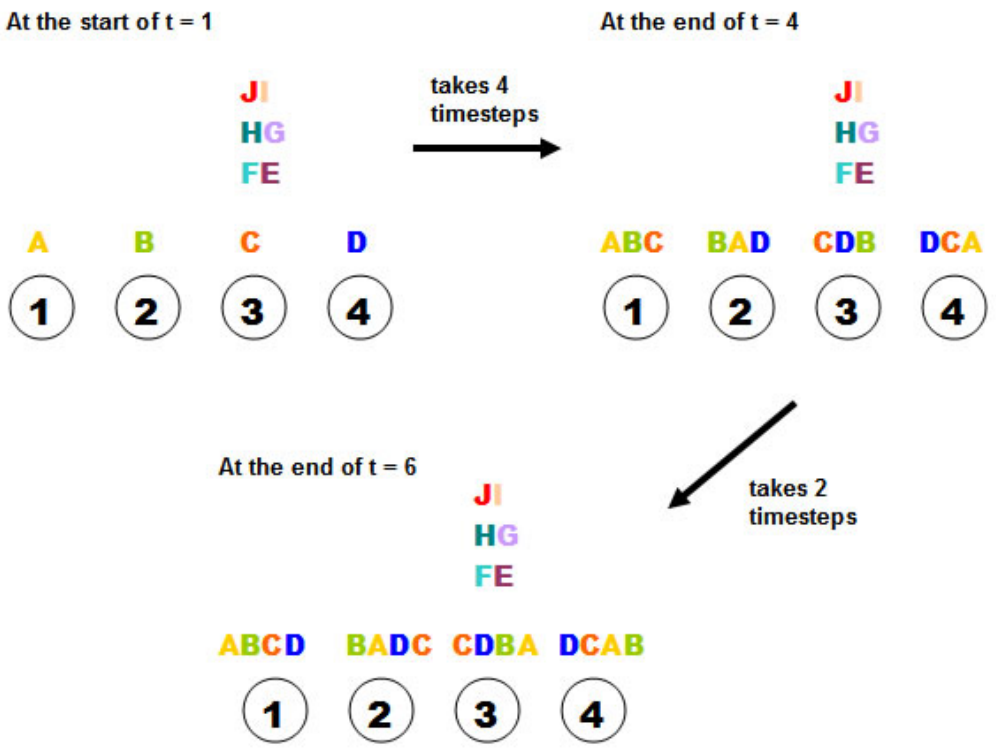

Figure 3.10: A distribution schedule for $N = 4$, $K = 4$, $L = 10$ that is not optimal but timesteps $t = 1$ until $t = 6$ are maximumly busy

Had the rarer file pieces from *Node 3* been replicated instead, the distribution process might have prevented from degrading into a *Single-Source Broadcasting* situation. In fact, this is exactly what is done in one instance of an optimal solution for the example. In the general algorithm we presented earlier in a previous section, we highlighted that giving replication priority to the current rarest file pieces in the network works fairly well in practice.

This counterexample shows that employing a greedy local maximizing busyness algorithm to maximize the amount of useful work done at a given timestep is simply insufficient. The selection and order of which file pieces to replicate also matters as this contributes towards ensuring that the network can remain *maximumly busy* in the succeeding timestep. We find that in general, giving replication priority to the current rarest file pieces works fairly well in ensuring *maximum busyness* in the next timestep.

3.5 Arbitrary Multi-source Broadcast

The *Arbitrary Multi-source Broadcast* problem is a more challenging extension of *Disjoint Multi-source Broadcast*. Here, the goal remains a broadcasting problem to distribute all L file pieces to all N nodes in a peer-to-peer data distribution system. Unlike *Disjoint Multi-source Broadcast* however, each of the initial K source nodes can hold overlapping subsets of the L file pieces, that is, each file piece may have more than one initial copy and can thus be held at more than one of the initial K sources at the start of the process. The set of initial sources for a file piece i, S_1^i where $1 \leqslant i \leqslant L$, is thus some arbitrary subset of the initial K sources, rather than restricted to a single node as in *Disjoint Multi-source Broadcast*. In *Arbitrary Multi-source Broadcast*, $1 < K \leqslant N$.

One of the current open problems we have identified about *Arbitrary Multi-source Broadcast* is whether the problem is NP-hard. Khuller et al. have proven that the more general data distribution problem, the *Data Migration Problem* is NP-hard, showing a polynomial-time reduction from the problem of edge coloring a simple graph with the smallest number of colors to an instance of the *Data Migration Problem* [10]. Under the *Data Migration Problem*, for every file piece i, where $1 \leqslant i \leqslant L$, both the set of initial source nodes for piece i and its set of destination nodes are some arbitrary subset of all N nodes. With the *Arbitrary Multi-source Broadcast* problem however, all L file pieces are desired by all N nodes but the initial source nodes for each of these file pieces are some arbitrary subset of the N nodes. It is this possible variance in the initial sources for each file piece that makes *Arbitrary Multi-source Broadcast* a challenging problem.

Despite being one of the variants of the broadcasting problem that we wanted to study initially for this thesis, due to the constraints of time, we were unable to make much progress for this particular problem. We nevertheless mark the *Arbitrary Multi-source Broadcast* case as the next step for future work, and we hope that our study of *Disjoint Multi-source Broadcast* have yielded us insights and ideas that would aid us in any future study of the *Arbitrary Multi-source Broadcast* problem.

Chapter 4

Conclusion and Future Work

The problem of broadcasting is itself a reasonable abstraction of the challenge of data distribution in peer-to-peer networks. By itself, it stands as a very interesting theoretical problem with many variations as we have seen and studied, from varying the initial conditions of the problem, such as the number of initial sources and initial copies of the file pieces, to modifying the underlying assumptions of the problem such as relaxing the *Half-Duplex*, *One-Port* or *1-Message* constraints.

In this paper, we have examined different variants of the broadcasting problem, and have specifically focused on *Disjoint Multi-source Broadcast* and *Arbitrary Multi-source Broadcast*, two variations that have not been studied much in past work from the area. We had derived a lowerbound on *Disjoint Multi-source Broadcast*, and presented a general algorithm and a set of best practices/heuristics that works well in practice for different instances of the problem. A few of the main heuristics that we identified are also employed in real-world peer-to-peer file distribution systems such as *BitTorrent*, which also gives replication priority to the current rarest pieces in the network in its piece selection algorithm, *Local Rarest First*.

We have also obtained findings that generalize to all variants of the broadcasting problem under the *Half-Duplex One-Port 1-Message* model. We identified a necessary condition for *maximum busyness* to exist in a given data distribution network (the most common *file piece set* must be at no more than $\lceil N/2 \rceil$ nodes). We found that employing a greedy local algorithm to maximize busyness at every timestep is insufficient towards creating a long-run optimal solution. Rather, the choice of which file pieces to replicate at each timestep also matters. This result reinforces the suitability of giving replication priority to the current rarest file pieces in the network.

One open problem that this paper has not been able to address is the development of an efficient algorithm to generate a *maximumly busy* configuration of file piece transfers during each timestep in Phase 2 and Phase 4 of the general algo-

rithm that we presented for *Disjoint Multi-source Broadcast*. The challenge for such an algorithm is determining which configuration from all possible *maximumly busy* configurations would assure that the resulting network remains *maximumly busy* in a non-terminal succeeding timestep. We have also been unable to address the *Arbitrary Multi-source Broadcast* case, and it remains an open problem if this variant of the broadcasting problem is NP-hard. We intend to begin any future work by starting with this particular problem.

We could also consider extensions to the *Half-Duplex One-Port 1-Message* model of the broadcasting problem, such as modifying the *Half-Duplex* constraint into a *Full-Duplex* assumption, allowing *n-Port* instead where $n > 1$ or permitting *P-Messages* on a connection where $P > 1$. In this paper, we have also assumed an underlying homogeneous peer-to-peer network where all link delays and switching times are equal to one unit. We might instead consider heterogeneous networks with different costs on different links and edges, as this assumption better reflects real-world peer-to-peer networks. Clearly, there are certainly many interesting avenues for future work that we can go from here.

Our study of *Disjoint Multi-source Broadcast* has yielded us insights into a very interesting theoretical problem and into the class of broadcasting problems as a whole. We have seen that studies and findings in broadcasting have the potential of benefiting a wide variety of real-world computer system problems that can be reasonably abstracted as a problem of broadcasting. We hope that the insights and ideas we have obtained from this study will aid us in any future work on other variants of broadcasting in peer-to-peer data distribution systems.

Bibliography

[1] Konstantin Andreev, Charles Garrod, Bruce M. Maggs, and Adam Meyerson. Simultaneous source location. In Klaus Jansen, Sanjeev Khanna, José D. P. Rolim, and Dana Ron, editors, *APPROX-RANDOM*, volume 3122 of *Lecture Notes in Computer Science*, pages 13–26. Springer, 2004.

[2] A. Bagchi, E. F. Schmeichel, and S. L. Hakimi. Parallel information dissemination by packets. *SIAM J. Comput.*, 23(2):355–372, 1994.

[3] Jean-Claude Bermond, Luisa Gargano, Adele A. Rescigno, and Ugo Vaccaro. Fast gossiping by short messages. In *Automata, Languages and Programming*, pages 135–146, 1995.

[4] A. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a bittorrent networks performance mechanisms. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–12, 2006.

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

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

[7] Z. Ge, D. R. Figueiredo, Sharad Jaiswal, J. Kurose, and D. Towsley. Modeling peer-peer file sharing systems. In *IEEE INFOCOM 2003: Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 2188–2198, March 2003.

[8] Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, and Xiaodong Zhang. Measurements, analysis, and modeling of bittorrent-like systems. In *IMC '05, 2005 Internet Measurement Conference*, pages 35–48, 2005.

[9] Samir Khuller, Yoo Ah Kim, and Azarakhsh Malekian. Improved algorithms for data migration. In Josep Díaz, Klaus Jansen, José D. P. Rolim, and Uri Zwick, editors, *APPROX-RANDOM*, volume 4110 of *Lecture Notes in Computer Science*, pages 164–175. Springer, 2006.

[10] Samir Khuller, Yoo Ah Kim, and Yung-Chun (Justin) Wan. Algorithms for data migration with cloning. *SIAM J. Comput.*, 33(2):448–461, 2004.

[11] Samir Khuller, Yoo Ah Kim, and Yung-Chun (Justin) Wan. On generalized gossiping and broadcasting. *J. Algorithms*, 59(2):81–106, 2006.

[12] Dejan Kostic, Ryan Braud, Charles Killian, Erik Vandekieft, James W. Anderson, Alex C. Snoeren, and Amin Vahdat. Maintaining high-bandwidth under dynamic network conditions. In *USENIX Annual Technical Conference, General Track*, pages 193–208. USENIX, 2005.

[13] R. Kumar and K.W. Ross. Peer assisted file distribution: The minimum distribution time. In *IEEE Workshop on Hot Topics in Web Systems and Technologies, HOTWEB 2006*, 2006.

[14] Arnaud Legout, G. Urvoy-Keller, and P. Michiardi. Rarest first and choke algorithms are enough. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 203–216, New York, NY, USA, 2006. ACM Press.

[15] Himabindu Pucha, David G. Andersen, and Michael Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In *Proc. 4th USENIX NSDI (to appear)*, Cambridge, MA, April 2007.

[16] Dongyu Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 367–378, New York, NY, USA, 2004. ACM Press.

[17] Krishna K. Ramachandran and Biplab Sikdar. An analytic framework for modeling peer to peer networks. In *INFOCOM*, pages 2159–2169. IEEE, 2005.

[18] Y. Tian, D. Wu, and K. W. Ng. Modeling, analysis and improvement for bittorrent-like file sharing networks. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–11, 2006.

[19] Niraj Tolia, Michael Kaminsky, David G. Andersen, and Swapnil Patil. An architecture for Internet data transfer. In *Proc. 3rd Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, May 2006.