ABSTRACT

Title of dissertation: ALGORITHMS FOR DATA DISSEMINATION AND COLLECTION

Yung-Chun Wan, Doctor of Philosophy, 2005

Dissertation directed by: Professor Samir Khuller
Department of Computer Science

Broadcasting and gossiping are classical problems that have been widely studied for decades. In broadcasting, one source node wishes to send a message to every other node, while in gossiping, each node has a message that they wish to send to everyone else. Both are some of the most basic problems arising in communication networks. In this dissertation we study problems that generalize gossiping and broadcasting. For example, the source node may have several messages to broadcast or multicast. Many of the works on broadcasting in the literature are focused on homogeneous networks. The algorithms developed are more applicable to managing data on local-area networks. However, large-scale storage systems often consist of storage devices clustered over a wide-area network. Finding a suitable model and developing algorithms for broadcast that recognize the heterogeneous nature of the communication network is a significant part of this dissertation.

We also address the problem of data collection in a wide-area network, which has largely been neglected, and is likely to become more significant as the Internet becomes more embedded in everyday life. We consider a situation where large amounts of data have to be moved from several different locations to a destination. In this work, we focus on two key properties: the available bandwidth can fluctuate, and the network may not choose the best route to transfer the data between two hosts.

We focus on improving the task completion time by re-routing the data through intermediate hosts and show that under certain network conditions we can reduce the total completion time by a factor of two. This is done by developing an approach for computing coordinated data collection schedules using network flows.

ALGORITHMS FOR DATA DISSEMINATION AND COLLECTION

by

Yung-Chun Wan

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2005

DEDICATION

To my parents

ACKNOWLEDGMENTS

I would like to take this opportunity to express my gratitude to all people who supported me and helped me making this thesis possible.

The first and foremost person I would like to thank is my advisor, Professor Samir Khuller, for his guidance and support. I thank him for first seeing my potential and talent ever since I was taking his course in my first semester at the University of Maryland. Since then he has been very patient with me and has been constantly providing me guidance. His door is always open, and I thank him for spending hours discussing problems with me. For each problem I worked on, he often suggested a few potential directions to tackle the problem. I also thank him for encouraging and generously supporting me to go to conferences and present papers there. He also encouraged me to go for summer internships, which have been a very valuable experience to me.

I am grateful to have a chance to work with Professor Leana Golubchik. She taught me how to do great system research and how to write good systems papers.

I thank Professors Samir Khuller, Bobby Bhattacharjee, Leana Golubchik, Mark Shayman, and Aravind Srinivasan for serving on my dissertation committee. I thank them for reading my thesis and giving suggestions. I also thank David Mount for serving on my preliminary examination committee.

I gratefully acknowledge funding support from NSF under the Awards CCR-9820965 and CCR-0113192 to Samir Khuller.

I would like to thank my mentors, including Michael R. Lyu, Jimmy Ho-Man Lee, John C.S. Lui, and Irwin King, in the Chinese University of Hong Kong. They helped me a lot and equipped me for the challenging graduate studies, and encouraged me to go abroad to pursue a Ph.D. degree. I also thank Connie Mansim Cheng, who inspired and encouraged me to come to the United States for further studies.

I am very fortunate to have to a chance to work with many dedicated fellow students. I learned a lot from Yoo-Ah Kim, who comes up with ideas quickly and thinks problems very carefully. Cheng-Fu Chou is always like a big brother to me. He made my transition to living in the States much easier. I also remember that we had great time together discussing and arguing research problems. I thank Rajiv Gandhi for encouraging me to work with Samir Khuller when I was a first-year Ph.D. student, and gave me many useful professional advice.

I am graceful to have wonderful friends during my studies. Without Sheung-Wah Ng, Chiu-Yuen Koo, Starsky Ho-Yin Wong, Nan Wang, Adam Woei-Jyh Lee, Sernam Lim, and Yingqian Zhang, I could only imagine I would have a dull life in Maryland. We worked hard and played hard, and I thank them for all their support.

Last but not the least, I would like to thank my family. I thank my brothers for their support for the family and taking care of everything while I was away from home. My deepest gratitude goes to my parents, for their unconditional love and support. My parents give all they have to my brothers and me. I dedicate this thesis to them.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

Broadcasting and gossiping are some of the most basic problems arising in communication networks. In this thesis, we are primarily concerned with problems related to broadcasting and gossiping that arise when managing large amounts of data. We study several different problems, all of which are related to data dissemination and collection on both local-area and wide-area networks.

The problems of broadcasting and gossiping have been widely studied for decades [66, 44, 48, 10, 11, 51]. The *broadcasting problem* is defined as follows: there are n nodes, and one source node needs to convey an item to every other node. In the *gossiping problem*, each node has an item that they wish to communicate to everyone else. We may treat the gossip problem as simultaneously performing n broadcasts. Communication is typically done in rounds, where in each round a node may send (or receive) an item to (or from) at most one other node. One typical objective function is to minimize the number of communication rounds. In this thesis, we use this objective function as the performance metric. Another typical objective function considered in the literature is to minimize the number of calls placed.

In the first part of this thesis we develop algorithms for problems that generalize broadcasting and gossiping. Large data storage systems store data on a collection of disks, connected via a fast local-area network. To deal with high demand for data, as well as for fault-tolerance, we may wish to have multiple copies of the same data item stored on a number of disks. Disks typically have constraints on storage and the number of clients that can simultaneously access data from it. A *data layout* specifies how many copies to have for each item, and which subset of disks to put it on. Given the demand for data items, computing a data layout that maximize the number of satisfied demands is NP-hard [86, 39]. Golubchik et al. [39] develop a polynomial time approximation scheme for this problem. Let us first consider a simple motivating example: suppose we initially have a collection of data items stored on a single disk (this could be tertiary storage) how do we create an initial layout? Each item has to be sent to an arbitrary subset of the disks. Moreover, we would like to create the layout as quickly as possible, because of the large amount of data involved. We call this problem the *single-source multicast problem.* (By *multicast*, we mean only a subset of nodes want an item.) This is a generalization of the *single-source broadcast problem* which has been solved optimally by Cockayne, Thomason, and Farley [23, 29]. In some cases, the data items are stored at different locations initially, and we still would like to create an initial layout. We call this problem the *multi-source multicast problem.* The *data migration problem* is a generalization of the above three problems, and constant factor approximation algorithm has been developed [59]. See Section 2.2 for details of this problem.

Another reason to study the broadcasting problem is because it is a primitive operation in many collective communication routines such as MPI (message-passing interface) [77, 40, 52]. Broadcasting arises when one wants to quickly distribute data to an entire network for processing. It is easy to see applications of single-source multicast and multi-source multicast problems in such systems. We will introduce the two problems and other generalizations in Section 1.2.1. Note that in some situations, when sending a small amount of data, latency is what is important. In other situations, when performing bulk transfers, bandwidth is important. By minimizing the number of communication rounds, we address both types of problems.

Many of the works on broadcasting and gossiping in the literature are focused on homogeneous networks. The algorithms developed are more applicable to managing data on local-area networks. However, large-scale storage systems often consist of storage devices distributed over a wide-area network, where data transfers over the wide-area network is much slower than data transfers through a local-area network. Therefore we wish to develop algorithms to create a data layout quickly over heterogeneous networks. We concentrate on studying the broadcasting problem in wide-area networks since it is the simplest problem, as well as the most basic one in the context of data dissemination. Another motivation for studying this problem is that broadcasting operations in many collective communication routines such as MPI [77] do not address the issue of optimizing the performance of communications on heterogeneous networks. For example, Networks of Workstations (NOWs) are a popular alternative to massively parallel machines and are widely used (for example the Condor project at Wisconsin [80] and the Berkeley NOW project [4]). By simply using off-the-shelf PC's, a very powerful workstation cluster can be created, and this can provide a high amount of parallelism at relatively low cost. With the recent interest in grid computing [33] there is an increased interest to harness the computing power of these clusters to have them work together to solve large applications that involve intensive computation. Several projects such as MagPIe [64] are developing platforms to allow applications to run smoothly by providing primitives for performing basic operations such as broadcast, multicast, scatter, reduce, etc. By recognizing and taking advantages of the heterogeneous nature of the underlying communication network, we may be able to implement broadcast faster. Some of the collective communication routines have been extended to clustered wide-area systems (see [64, 63, 15]). However, many of these primitives are implemented using simple heuristics without approximation guarantees. In the second part of this thesis we are interested in developing suitable model and understanding the difficult issues and challenges in implementing broadcasting and multicasting over clustered wide-area networks.

In the third part of this thesis we are interested in a data collection problem. Data collection in wide-area network, which has largely been neglected in the past, is likely to become more significant as the Internet becomes more embedded in everyday life. In particular, large-scale data collection problems correspond to a set of important *upload* applications. These applications include online submission of income tax forms, submission of papers to conferences, submission of proposals to granting agencies, Internet-based storage, and many more. Another example of a data collection problem is for high-performance computing applications where large amounts of data need to be transferred from one or more data repositories to one or more destinations, where computation on that data is performed. One more example is data mining applications where large amounts of data may need to be transferred to a particular server for analysis purposes. One characteristic of data collection that differs from data dissemination is that the destination node has to receive all the data, while in a data dissemination problem, other nodes may send data items for the source node. Using a graph-theoretic formulation, we develop a data collection algorithm that returns a *coordinated* data collection schedule which would afford maximum possible utilization of available network resources. Due to the discrepancy between our graph-theoretic abstraction and the way a TCP/IP network works on the Internet, we also present a comprehensive study which compares the performance, robustness, and adaptation characteristics of three potential approaches to the problem.

We often like to complete the data dissemination or collection process as soon as possible for various reasons. However, the communication network between storage devices is usually the bottleneck in transferring data, especially in such large-scale settings in the motivating examples mentioned above. It is critical to find efficient algorithms to schedule the transfer of data to fully utilize the network. We can model these scheduling problems as combinatorial optimization problems. Unfortunately many of these problems are hard to solve *optimally.* We classify a problem as hard to solve if it is known to be NP-hard. An algorithm is *efficient* if its worst-case running time is bounded by a polynomial function of its input size. Under the widely believed conjecture of $P \neq NP$, efficient algorithms to find the optimal solution, do not exist for NP-hard problems. Most of the interesting data dissemination problems are at least of moderate sizes. Therefore it takes too much time to run inefficient algorithms to obtain the optimal solution. It is then natural not to opt for getting the optimal solution, but to settle for obtaining efficient algorithms that may not always return the optimal solution. Fortunately, there exist efficient algorithms that are guaranteed to produce solutions that are within a certain factor of the optimal solution. This class of algorithms is called *approximation algorithms*, and is often used for NP-hard optimization problems, because it trades quality for tractability. This class of algorithms was formally introduced by Garey, Graham, and Ullman [36] and Johnson [54]. Johnson noted that although all NP-complete optimization problems are equally hard to find the optimal solution, close-to-optimal solutions are easier to find for some problems. Approximation algorithms have been an active field of research over the past two decades. Many works have been done on developing techniques to prove approximation guarantees, and some works have been devoted to show that approximating a problem better than a certain factor is impossible unless P = NP. Books by Hochbaum [45] and Vazirani [89] provide good starting points for further readings on these topics.

There are other ways to evaluate the performance of an algorithm. While approximation algorithms provide a guarantee on the worst-case performance of an algorithm, we can evaluate the typical performance of an algorithm by running simulations using typical inputs. Problems under sophisticated models may be too difficult to analyze mathematically, and thus experimental evaluation allows us to evaluate algorithms running on much more complex systems using more sophisticated models. Moreover, experimental evaluations can be used to compare the typical performance of an approximation algorithm to its worst-case guarantee. This often gives us an idea as to whether the upper bound on the performance of the algorithm and the lower bound on the optimal solution are tight or not. Thus it provides us an insight on how to improve the analysis of the algorithm.

We want to emphasize that these two performance evaluation techniques are complementary. In this thesis, both techniques are used where appropriate.

1.1 Preliminaries

The formal definition of an approximation algorithm is as follows.

Definition 1.1.1 *A ρ-approximation algorithm, $\mathcal{A}$, for a minimization problem, is a polynomial-time algorithm, such that for every valid instance I, $\mathcal{A}$ returns a solution of cost at most $\rho \cdot OPT(I)$, where $\rho > 1$ and $OPT(I)$ denotes the objective function value of an optimal solution to instance I (we will shorten this to OPT). The ratio ρ is called the approximation ratio or the approximation guarantee of the algorithm.*

A general approach in obtaining an approximation algorithm is to find in polynomial time a *lower bound* to the cost of the an optimal solution. For the problems considered in this thesis, we compute lower bounds by exploiting the combinatorial structure of the problems. If the upper bound on the worst-case performance of our algorithm is within ρ times the lower bound, we have a ρ-approximation algorithm.

A *polynomial time approximation scheme* (PTAS) for a problem is an algorithm that, for each fixed $\epsilon > 0$, gives a $(1+\epsilon)$-approximation algorithm with running time polynomial in the size of the input (but maybe exponential in $1/\epsilon$). Therefore we can compute a solution arbitrarily close to the optimal solution.

Makespan is used to measure the performance of many scheduling problems, and is defined as follows.

Definition 1.1.2 *By makespan, we refer to the time it takes to finish all data transfers for data dissemination and collection problems.*

1.2 Organization and Overview of Contributions

This thesis is organized in eight chapters. We survey the work related to the research presented in this thesis in Chapter 2. In Chapters 3 to 6 we describe several data dissemination problems that generalize broadcasting and gossiping problems. We develop approximation algorithms to solve these problems. In Chapter 7 we describe a coordinated data collection problem. Since the communication model is quite involved, we evaluate the performance of our algorithms by simulations. We conclude and describe future work in Chapter 8.

We now describe the motivations and contributions of this thesis in the remainder of this chapter.

1.2.1 Data Dissemination Problems

We consider several data dissemination problems which are generalizations of broadcasting and gossiping problems under a local-area network. We then study the broadcasting problem under a more complex model, which tries to model characteristics of a wide-area network. The difficulty of the broadcasting and gossiping problems depends on the assumptions in the underlying communication model. We first describe a few models commonly found in the literature, then we describe the models we use, the problems we consider, and our results.

Different underlying communication graphs are considered in the literature. Bumby [16] developed an optimal algorithm for the gossiping problem by assuming the underlying communication graph is complete, i.e., any two nodes can directly communicate with one another. Bar-Noy et. al. [8] assume the communication graph is complete with arbitrary costs on the edges. This model allows different pairs of nodes to communicate at different costs, but it makes the problem much harder. They have a polynomial-time $O(\log n)$-approximate algorithm, which involves solving a linear program, for the broadcasting problem. Moreover, arbitrarily connected graphs have been considered, i.e., only adjacent nodes in the graph can communicate. Elkin-Kortsarz [28] give a polynomial-time $O(\frac{\log n}{\log \log n})$-approximation algorithm, which also involves solving a linear program, for the broadcasting problem. Special graphs, for example, trees [87], grids [30, 31], hypergraphs [81, 69], and directed graphs [43], are also considered.

Besides the communication graph, different works assume different ways to exchange items. Communication is typically done in rounds; in the *telephone model* [41], a node may communicate with at most one other node in each call (and each call takes one round). This is in contrast of having a broadcast channel, where one node may inform several other nodes in one round. There are two sub-models of interest. The *half-duplex model* [67, 44, 34] allows a node to send (receive) some information to (from) one other node in a single round. The *full-duplex model* [67, 44, 34] allows each node to participate in at most one call in each round, and allows two nodes to exchange all the information they know in each call. Note that both sender and receiver are busy during the transfer of information. Conveying an item from one node to another can be viewed as sending a message. In the *postal model* [9], each message simply has a latency of C time units when the message is sent from one node to another. The sender is busy for only one time unit while the message is being injected into the network. The message takes C time units of transit time and the receiver is busy for one time unit when the message arrives. This send-and-forget nature of passing messages with communication latencies essentially captures the communication pattern as discussed in several papers that deal with implementations of systems to support such primitives (see [64, 63]). The *LogP model* [24] is a more realistic model where it takes both latency and throughput of the underlying network into account.

In the past most work has been concentrated on allowing arbitrarily large message size, meaning that two nodes may exchange all the information that they have in one round. However, in the gossiping problem, the message size may grow with the number of nodes. If each piece of data item is large, and the cost of setting up a call is small when compared to the cost of transferring an item, a more suitable model is to assume one can exchange only a single item in a call (i.e., in constant time). Bermond et al. [10, 11] considered this *"short messages" model*, and give an optimal algorithm for the gossiping problem.

We now describe two of our works which are related to broadcasting and gossiping.

Generalized Broadcasting and Gossiping

As stated at the beginning of the introduction, this part of our work is motivated by the problem of creating initial data layouts in parallel disk systems. This problem can be treated as generalizations of the broadcasting and gossiping problem. Each node models a *disk* in the system, and a *data item* needs to be transferred to a set of disks. If each disk had exactly one data item, and needs to copy this data item to every other disk, then it is exactly the problem of gossiping.

The communication model we use is the half-duplex telephone model, where only one data item may be communicated between two communicating nodes during a single round. Each node may communicate (either send or receive an item of data) with at most one other node in a round. This model, the same as in the work by [42, 3], best captures the connection of parallel storage devices that are connected on a local-area network and is most appropriate for our application. Note that this is also a "short messages" model because only one data item can be transferred in one round.

The basic generalizations of gossiping and broadcasting problems that we are interested in are of two kinds: (a) each data item needs to be communicated to only a subset of the nodes, and (b) several data items may be known to one node. Similar generalizations have been considered before by Liben-Nowell [72], and Richards and Liestman [81], but they assume transferring long messages is allowed. In Section 2.2 we discuss in more detail the relationships between our problem and the ones considered in those papers.

Suppose we have N nodes and Δ data items. The problems we are interested in are:

1. *Single-source multicast.* There are Δ data items stored on a single node (the source). We need to send data item i to a specified subset D_i of nodes. Figure 1.1 shows the initial and target layouts, and their corresponding D_i's for a single-source multicast instance when Δ is 4.

2. *Multi-source broadcast.* There are Δ data items, each stored separately at a single node. These need to be broadcast to all nodes. We assume that data item i is stored on node i, for $i = 1 \ldots \Delta$.

3. *Multi-source multicast.* There are Δ data items, each stored separately at a single node. Data item i needs to be sent to a specified subset D_i of nodes. We assume that data item i is stored on node i, for $i = 1 \ldots \Delta$.

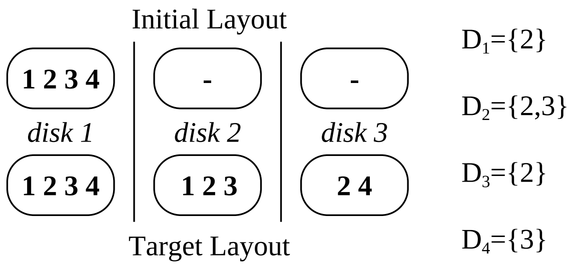


Figure 1.1: An example of a single-source multicast instance.

Considering the multicast problems, we use no bypass (intermediate) nodes as holding points for the data, i.e., we move data only to nodes that need the data. However, bypass nodes can be used to hold data temporarily so that we can finish the multicasting in a shorter time. For example, a source for item i may send i to a bypass node, which later forwards the item to a node in D_i even though the bypass node itself is not in D_i.

One potential concern with the communication model is that it allows an arbitrary number of nodes to communicate simultaneously in each round. This is of concern if the network connecting the nodes do not have enough capacity to permit such arbitrary communication patterns. We may restrict the *total* number of simultaneous transfers that may be going on in each round. We call this model *bounded-size matching model.*

In joint work with S. Khuller and Y. Kim [61, 58], we develop algorithms for these three problems. We summarize our results as follows.

1. In Chapter 3 we discuss the single-source multicast problem and give a polynomial-time algorithm that outputs a solution where the number of rounds is at most $OPT + \Delta$.

2. In Chapter 4 we discuss the multi-source broadcast problem and give a polynomial-time algorithm that outputs a solution where the number of rounds is at most $OPT + 3$. In particular the number of rounds needed is $\lceil \log \frac{N}{\Delta} \rceil + 2\Delta$.

3. In Chapter 5 we discuss the multi-source multicast problem and prove that the problem is NP-hard. We give a polynomial-time algorithm that outputs a solution where the number of rounds is at most $4OPT + 2$. Then we present an improved algorithm that outputs a solution where the number of rounds is at most $(3 + o(1))OPT$. We also present a 3-approximation algorithm for the special case in which the source disks are not in any subset D_i. If bypass nodes are allowed, we obtain a polynomial-time algorithm that outputs a solution where the number of rounds is at most $3OPT + 6$.

4. In Section 5.5 we develop a method to convert any constant factor approximation algorithm for the full matching model to a constant factor approximation algorithm for the bounded-size matching model. The only constraint is that no bypass disks are allowed. Therefore, we obtain constant factor approximation algorithms for all of the above three problems.

Broadcasting in Two-tier Communication Networks

As mentioned at the beginning of the introduction, we are interested in developing suitable models, and understanding the difficult issues and challenges in implementing broadcast and multicast operations on systems that run on clustered wide-area networks. It has applications in Networks of Workstations and grid computing.

To model a heterogeneous network, one may want to change the model of the underlying communication graph in the broadcasting problem. Many different underlying communication graphs have been considered in the literature. For example, Elkin-Kortsarz [28] considered arbitrarily connected graphs, with the property that only nodes adjacent in the graph may communicate. However, this model is too restrictive and allows direct communication only between nodes adjacent in a certain communication graph. The postal [9] and LogP [24] models take user-specified parameters such as latency and throughput into account to better model the underlying communication network. However, broadcast algorithms proposed for both models are running on homogeneous networks and thus not well suited for our applications. Therefore we need a communication model which allows some pairs of nodes to communicate quicker than some other pairs. Various models for heterogeneous environments have been proposed in the literature. One general model is the one proposed by Bar-Noy et al. [8] where the communication costs between links are not uniform. In addition, the sender may engage in another communication before the current one is complete. An approximation algorithm with a guarantee of $O(\log k)$ is given for the operation of performing a multicast of size k. However, the algorithm is complicated and involves solving a linear program. See Section 2.3 for descriptions of other models for heterogeneous networks.

Consider several clusters of workstations. Each local cluster (sometimes this is also called a subnet) is connected on a fast local area network, and inter-cluster communication is via a wide-area network. In such situations, the time taken for a pair of nodes in the same cluster to communicate, can be significantly smaller than the communication time of a pair of nodes in different clusters. In fact, in the work by Lowekamp and Beguelin [74] they also suggest methods for obtaining the subnets/clusters based on communication delays between pairs of nodes.

Motivated by this, the *communication model* we consider is the following. There are k clusters of nodes. Cluster i has size n_i. We will assume that in one time unit, a node can send a message (or a data item) to any one node in its own cluster. However, sending a message from one node to another node in a different cluster takes C time units. Even if the nodes in a cluster are heterogeneous, their transmission times are usually much less than the communication time across clusters. We also assume that a node can be sending or receiving a message from only one node at any point of time (a *matching*). We believe that this model well captures the heterogeneous nature of clustered wide-area systems, yet simple enough for us to design efficient algorithms with good approximation ratio. In this model Lowekamp and Beguelin [74] propose some simple heuristics for performing broadcast and multicast. However, these heuristics may produce solutions that are arbitrarily far from optimal.

Similar to the model used in generalized broadcasting and gossiping, one potential concern with the above two-tier communication model is that it allows an arbitrary number of nodes to communicate simultaneously in every time step. This is of concern if the global network connecting the different clusters does not have enough capacity to permit such arbitrary communication patterns. There are several ways in which we can restrict the model. One model that we propose is the *bounded degree model* where each cluster i is associated with a parameter d_i that restricts the *number* of nodes from this cluster that can communicate with nodes outside this cluster in each time step. Another possible manner is to use the *bounded-size matching model.* We restrict the *total* number of simultaneous *global* transfers that may be going on in each time step without restricting the number of transfers into/out of a single cluster.

In addition, we consider the postal model [9] version where each message simply has a latency of C time units when the message is sent from one node to another node belonging to a different cluster. The sender is busy for only one time unit while the message is being injected into the network. The message takes C units of transit time and the receiver is busy for one unit of time when the message arrives. This model essentially captures the communication pattern as discussed in several papers that deal with implementations of systems to support such primitives (see [64, 63]).

In joint work with S. Khuller and Y. Kim [60], we develop broadcasting and multicasting algorithms in two-tier communication networks. We summarize our results as follows.

1. We propose Algorithm *LCF* (Largest Cluster First) for performing broadcasting in the basic two-tier model, and develop multicasting algorithm based on *LCF*. We show that these

algorithms produce solutions where the makespan is not more than the optimal by a factor of 2. Moreover, the analyses are tight for both algorithms.

2. For the bounded degree model we show how to reduce the broadcasting and multicasting problems to instances of the basic model to develop algorithms with approximation ratio of 3 for both problems.

3. For the bounded-size matching model we develop algorithms with approximation ratio of 2, using algorithms in the basic model, for both broadcasting and multicasting.

4. For the postal model, Algorithm *LCF* gives a factor 3 approximation. In addition, we present another algorithm, called *Interleaved LCF*, and show that the makespan is at most 2 times OPT' where OPT' is the minimum makespan among schedules that minimize the total number of global transfers.

One issue with our broadcasting protocol is that it assumes knowledge of the sizes of the clusters. In some applications, the cluster sizes may not be known accurately in advance. In the basic two-tier model, we study the effect of having inaccurate information regarding the sizes of the clusters to the *LCF* algorithm in Section 6.6. In fact, as we demonstrate, even if there is a multiplicative factor of 2 inaccuracy in the sizes of the clusters, there is hardly any change in the performance of *LCF*.

We will describe these works in Chapter 6.

1.2.2 Coordinated Data Collection

As mentioned at the beginning of the introduction, we are interested in developing algorithms for large-scale data collection in wide-area network. In the data collection problem, we have a set of source hosts, each stores some amount of data. We would like to find a transfer schedule which minimizes the makespan to collect all data to a given destination host. Using a graph-theoretic formulation, we develop data collection algorithm that returns a *coordinated* data collection schedule which would afford maximum possible utilization of available network resources. One difficult issue is due to the discrepancy between our simple graph theoretic abstraction and the way a TCP/IP network works on the Internet. For example, in the graph-theoretic abstraction we assume the available bandwidth between a pair of hosts has a fixed value. However, the background traffic on the Internet may change dynamically and so a network may not be able to sustain the same amount of traffic at a later time. Moreover, we may want to take into account the slow start behavior of Transmission Control Protocol (TCP) and possible shared congestion links in scheduling transfers. The available bandwidth may also depend on the number of TCP connections connected to other hosts at that time. Modeling these properties of TCP exactly is extremely complex. Therefore, we evaluate our algorithms by simulations.

We study *application-level* approach to improve performance of a large-scale data collection problem (from multiple source hosts to a destination server) in the context of an upload architecture *Bistro*. *Bistro* is a scalable and secure *application-level* architecture for wide-area upload applications [13]. Hosts which participate in this architecture are termed *bistro*s. Given a large number of clients that need to upload their data to a given destination server, Bistro breaks the upload process into three steps: (1) a timestamp step to ensure that the data is submitted on-time, for applications with deadlines, *without* having to actually transfer the data, (2) a data transfer step, where clients *push* their data to bistros, and (3) a data collection step, where the destination server *pulls* data from bistros. We note that during step (2) receipts corresponding to clients' transfers are sent by the bistros to the destination server; hence the destination server knows where to find all the data which needs to be collected during step (3). We also note that in this context there are no deadline issues, as any deadlines associated with an upload application are taken care of in step (1) above.

Consequently, our *data collection problem* can be stated as: *Given* a set of source hosts, the amount of data to be collected from each host, a common destination host for the data, and available link capacities between hosts, *our goal is to* construct a data transfer schedule which specifies on which path, in what order, and at what time should each "piece" of data be transferred to the destination host, *where the objective is to* minimize the time it takes to collect all data from the source hosts, i.e., makespan. Since we are focusing on application-level solutions, a path (above) is defined as a sequence of hosts, where the first host on the path is the source of the data, intermediate hosts are other bistros (hosts) in the system, and the last host on the path is the destination host. The transfer of data between any pair of hosts is performed over TCP/IP, i.e., the path the data takes between any pair of hosts is determined by IP routing.

There are, of course, simple approaches to solving the data collection problem; for instance: (a) transfer the data from all source hosts to the destination host in parallel, or (b) transfer the data from the source hosts to the destination host sequentially in some order, or (c) transfer the data in parallel from a subset of source hosts at a time, and possibly during a predetermined time slot, as well as other variants (refer to Section 7.2 for details). We refer to these methods as *direct*, since they send data directly from the source hosts to the destination host. However, long transfer times between one or more of the hosts (holding the data) and the destination server can significantly prolong the amount of time it takes to complete a large-scale data collection process. Such long transfer times can be the result of poor connectivity between a pair of hosts, or it can be due to wide-area network congestion conditions, e.g., due to having to transfer data over one or more peering points whose congestion is often cited as cause of delay in wide-area data transfers [71]. Given the current state of IP routing, congestion conditions may not necessarily result in a change of routes between a pair of hosts, even if alternate routes exist.

Another approach to dealing with such congestion problems might be to use application-level re-routing techniques as we stated above. However, we believe that in the case of a large-scale data collection problem, the issue is not only to avoid congested link(s), but to devise a *coordinated* transfer schedule which would afford maximum possible utilization of available network resources between multiple sources and the destination. (We formulate this notion more formally below.)

Given the above stated data collection problem, additional possible constraints include (a) ability to split chunks of data into smaller pieces, (b) ability to merge chunks of data into larger pieces, and (c) storage related constraints at the hosts. To focus the discussion, we consider the following constraints. For each chunk of data we allow (a) and (b) to be performed only by the source host of that data and the destination host. We also do not place storage constraints on hosts but rather explore storage requirements as one of the performance metrics.

In joint work with C. Chou, W. Cheng, L. Golubchik, and S. Khuller [19], we propose an approach for computing coordinated data collection schedules. We also present a comprehensive study which compares the performance, robustness, and adaptation characteristics of the three potential approaches to large-scale data transfers in IP-type networks, namely direct, non-coordinated, and coordinated approaches. We do this using ns2 [50] simulations and within the context of our graph-theoretic model. (The specific performance metrics used in this comparison are defined in Section 7.5.) This study (also in our previous work [21]) shows that coordinated methods can perform better than non-coordinated and direct methods under various degrees and types of network congestion. These improvements are achieved under low storage requirement overheads and without significant effects on other network traffic throughput. In addition, the study (also in our previous work [22]) shows that coordinated methods are more robust than non-coordinated methods under inaccuracies in network condition information. Furthermore, we compare the adaptation characteristics of the coordinated vs. non-coordinated methods under changes in network conditions. These changes occur after the data transfer schedule is computed and while the corresponding data transfer is already in progress. An adaptation to network conditions study is important because the above stated applications are long lasting (i.e., correspond to large data transfers). Hence, it is very likely that network conditions will change while the data transfer process is in progress. Our adaptation study shows that the coordinated approach has a greater potential for adaptation than a non-coordinated method and hence results in better performing data transfers.

Chapter 2

Related Work

In this chapter, we give a brief literature survey of research related to the work in this thesis. We first discuss previous work on how to compute a data layout, because a data layout forms input instances of several of the data dissemination problems considered in this thesis. We then discuss related work on broadcasting and gossiping problems, broadcasting in two-tier communication networks, and coordinated data collection.

2.1 Computing Data Layout

One motivating example of our work on generalizations of broadcasting and gossiping is the *data placement problem*, which arises in the context of storage systems. In a homogeneous storage system, there are several disks. Each disk has the same storage capacity, and the same load capacity, the maximum number of clients that it can serve simultaneously. Given a number of data items and the number of clients requesting the items, the goal is to find a placement of items on disks so as to maximize the total number of clients satisfied, subject to the capacity constraints.

Golubchik et al. [39] develop an algorithm that can always pack a $(1 - \frac{1}{(1+\sqrt{k})^2})$-fraction of items for any instance of homogeneous storage systems, where k is the storage capacity of a disk. They also give a polynomial time approximation scheme for the problem. They also consider a variant of the problem, called uniform ratio storage systems, where disks are not identical. Instead, the problem only requires a uniform ratio of the load to the storage capacity is identical for each disk. They develop an algorithm that can always pack a $(1 - \frac{1}{(1+\sqrt{C_{min}})^2})$-fraction of items, where C_{min} is the minimum storage capacity of any disk. They also give a polynomial time approximation scheme for this problem. Kashyap and Khuller [56] studied the data placement problem for homogeneous storage systems where data items may have different sizes, and give a polynomial time approximation scheme.

2.2 Generalized Broadcasting and Gossiping

The simplest problem under our model in generalized broadcasting and gossiping is the *single-source broadcast* problem. There are Δ data items stored on a single node (the source). We need to broadcast all items to all $N-1$ remaining nodes. This problem was solved optimally by Cockayne, Thomason and Farley [23, 29]. The schedule takes $2\Delta - 1 + \lfloor \log N \rfloor$ rounds for odd N and $\left\lceil \frac{\Delta(N-1)-2^{\lfloor \log_2 N \rfloor}+1}{\lfloor N/2 \rfloor} \right\rceil + \lfloor \log N \rfloor$ rounds for even N.

One general problem of interest is the *data migration problem* when data item i resides in a specified (source) subset S_i of nodes, and needs to be copied to a (destination) subset D_i. This problem is more general than multi-source multicast problem where we assumed that $|S_i| = 1$ and that all the S_i's are disjoint. For the data migration problem we have developed a 9.5-approximation algorithm [59]. While this problem is a generalization of single-source broadcast, multi-source broadcast, and the multi-source multicast problem (and clearly also NP-hard since even the special case of multi-source multicast is NP-hard), the bounds developed in [59] are not as good as the bounds we obtain for the specific problems. The methods used for single-source multicast and multi-source broadcast are completely different from the algorithm in [59]. Using the methods in [59] one cannot obtain additive bounds from the optimal solution. The algorithm for multi-source multicast presented in Chapter 5 is a simplification of the general algorithm developed [59], and we obtain a much better approximation factor of 4. By using new ideas we can improve it to $3 + o(1)$.

Hall et al. [42] studied a special case of the data migration problem. Given a specified subset of nodes in which each data item resides, and a set of move operations, where each move operation specifies which data item needs to be moved from one node to another, the problem is to schedule these move operations to minimize makespan. Without space constraints, this problem is can be reduced to edge-coloring on a multigraph. With space constraints, they give approximation algorithms using bypass nodes and without bypass nodes.

Many generalizations of gossiping and broadcasting have been studied before. For example, the paper by Liben-Nowell [72] considers a problem very similar to multi-source multicast with

$\Delta = N$. However, the model that he uses is different than the one that we use. In his model, in each telephone call, a pair of nodes can exchange all data items that they know (i.e., long messages). The objective is to simply minimize the total number of phone calls required to convey data item i to set D_i of nodes. In our case, since each data item might take considerable time to transfer between two nodes, we cannot assume that an arbitrary number of data items can be exchanged in a single round (i.e., we allow only short messages). Several other papers use the same telephone call model [6, 16, 41, 51, 88]. Liben-Nowell [72] gives an exponential time exact algorithm for the problem.

Other related problem that has been studied is the set-to-set gossiping problem [70, 81] where we are given two possibly intersecting sets A and B of nodes and the goal is to minimize the number of calls required to inform all nodes in A of all the items known to members in B. The work by Lee and Chang [70] considers minimizing both the number of rounds as well as the total number of calls placed. The main difference is that in a single round, an arbitrary number of items may be exchanged. For a complete communication graph they provide an exact algorithm for the minimum number of calls required. For a tree communication graph they minimize the number of calls or number of rounds required. Liben-Nowell [72] generalizes this work by defining for each node i the set of relevant items that they need to learn. This is just like our multi-source multicast problem with $\Delta = N$, except that the communication model is different, as well as the objective function. The work by [76] also studies a set-to-set broadcast type problem, but the cost is measured as the total cost of the broadcast trees (each edge has a cost). The goal is not to minimize the number of rounds, but the total cost of the broadcast trees. In [34] they also define a problem called scattering which involves one node broadcasting distinct messages to all the other nodes (very much like our single-source multicast, where the multicast groups all have size one and are disjoint).

2.3 Broadcasting in Two-tier Communication Networks

Broadcasting efficiently is an essential operation and many works are devoted to this under a number of communication models (see [81, 44, 55, 9, 12] and references therein). For example, Elkin-Kortsarz [28] consider minimizing the broadcast time in arbitrarily connected graphs, with the property that only adjacent nodes in the graph may communicate. However, the approximation guarantee is $O(\frac{\log n}{\log \log n})$. The postal model [9] captures the communication latency when passing a message, and optimal broadcast algorithm was developed. The LogP model [24] suggests an alternative framework when dealing with nodes in a single cluster, and it captures both the communication latency and throughput in the network. Broadcasting algorithms [55] for the LogP model have been developed and shown to be optimal. However, all algorithms under the above models only work under homogeneous environment.

Various models for heterogeneous environments have been proposed in the literature. One general model is the one proposed by Bar-Noy et al. [8] where the communication costs between links are not uniform. In addition, the sender may engage in another communication before the current one is complete. An approximation algorithm with a guarantee of $O(\log k)$ is given for the operation of performing a multicast of size k. Another simple model for heterogeneous networks of workstations was proposed by Banikazemi et al. [7]. In this model, heterogeneity among processors is modeled by a non-uniform speed of the sending processor. A heterogeneous cluster is defined as a collection of processors $p_1, p_2, \ldots, p_n$ in which each processor is capable of communicating with any other processor. Each processor has a transmission time which is the time required to send a message to any other processor in the cluster. Thus the time required for the communication is a function of only the sender. Each processor may send messages to other processors in order, and each processor may be receiving only one message at a time. They proposed a simple heuristic called the Fastest Node First (FNF) heuristic, which was studied further by Liu [73] and by Khuller and Kim [57, 62]. However, in this model it is assumed that the time taken by a processor to send a message to *any* other processor is the same. This is the main limitation of the model.

Lowekamp and Beguelin [74] considered the same two-tier communication network model as in our work in Chapter 6. However, the heuristics they proposed for broadcasting may produce solutions arbitrarily far from optimal.

2.4 Coordinated Data Collection

Many works have been done to provide ways to deliver data by not following the default network route. Although some works exist on multipoint-to-point aggregation mechanisms at the IP layer [5, 17], such solutions have focused on reduction of overheads due to small packets (e.g., ACKs) and usually require the use of an active networks framework which is not currently widely deployed over the public Internet. Another approach is application-level re-routing, which is used to improve end-to-end performance, or provide efficient fault detection and recovery for wide-area applications. For instance, in [84] the authors perform a measurement-based study of comparing end-to-end round-trip time, loss rate, and bandwidth of default routing vs alternate path routing. Their results show that in 30% to 80% of the cases, there is an alternate path with significantly superior quality. Their work provides evidence for existence of alternate paths which can outperform default Internet paths. Other frameworks or architectures which consider re-routing issues include Detour [83] and RON [2]. The Detour framework [83] is an informed transport protocol. It uses sharing of congestion information between hosts to provide a better "detour path" (via another node) for applications to improve the performance of each flow and the overall efficiency of the network. This work also provides evidence of potential long-term benefits of "detouring" packets via another node by comparing the long-term average properties of detoured paths against default Internet paths. The Resilient Overlay Network (RON) [2] is an architecture allowing distributed Internet applications to detect failure of paths (and periods of degraded performance) and recover fairly quickly by routing data through other (than source and destination) hosts. It also provides a framework for implementing expressive routing policies. The above mentioned re-routing schemes focus on architectures, protocols, and mechanisms for accomplishing application-level re-routing through the use of overlay networks. They provide evidence that such approaches can result in significant performance benefits. We consider a similar environment (i.e., application-level techniques in an IP-type wide-area network) in Chapter 7. However, an important distinction is that previous works (except ours [21, 22]), do not consider *coordination* of multiple data transfers. All data transfers are treated independently, and each takes the “best” application-level route available. We refer to such techniques as “non-coordinated” data transfers. In contrast, our goal is to construct application-level *coordinated* data transfers.

Chapter 3

Single-Source Multicasting

In this chapter[1], we consider *single-source multicasting problem*, where there is one source disk s that has all Δ items and others do not have any item in the beginning, and we would like to send item i to disks in set D_i. We develop an algorithm where D_i can be an arbitrary subset of disks. The number of rounds required by our algorithm is at most $\Delta + OPT$ where OPT is the minimum number of rounds required for this problem. Our algorithm is obviously a 2-approximation for the problem, since Δ is a lower bound on the number of rounds required by the optimal solution. We may apply this algorithm to create a new data layout from one server to servers at different locations, where each item has to distribute to different subset of servers.

3.1 Problem Specification

Suppose we have N disks and Δ data items. The single-source multicast problem is defined as follows:

Single-source multicast. There are Δ data items stored on a single disk (the source). We need to send data item i to a specified subset D_i of disks. Figure 1.1 shows the initial and target layouts, and their corresponding D_i's for a single-source multicast instance when Δ is 4. Our goal is to find a schedule using the minimum number of rounds, that is, minimizing the makespan, subject to the following communication model.

3.1.1 Model

We assume that the underlying network is complete and the data items are all of the same size; in other words, it takes the same amount of time to migrate an item from one disk to another. The crucial constraint is that each disk can participate in the transfer of only one item—either

[1]This is joint work with S. Khuller and Y. Kim [61, 58].

as a sender or receiver. This is the same model as in the work by [42, 3] where the disks may communicate on any matching. For example, *Storage Area Networks* support a communication pattern that allows for devices to communicate on a specified matching. Moreover, as we do not use any bypass disks, all data is only sent to disks that desire it.

Note that after a disk receives item i, it can be a source of item i for other disks that have not received the item as yet.

3.2 Algorithm Single-Source Multicast

Our algorithm consists of two phases. In the first phase, we make exactly $\lfloor |D_i|/2 \rfloor$ copies for all items i. Once each item i has $\lfloor |D_i|/2 \rfloor$ copies, in second phase we can finish migrating one item at each round by copying from the current copies to the remaining $\lfloor |D_i|/2 \rfloor$ disks in D_i which have not received item i as yet and using the source disk to make another copy if $|D_i|$ is odd.

It is easy to see that the second phase can be scheduled without conflicts as we deal with only one item in each round. For the first phase, let us consider the simple example shown in Figure 3.1. In this example, all D_i are identical and include all disks (thus the problem is the same as single-source broadcast [23, 29]) and $\Delta = 4$, $|D_i| = 12$ for each item i. At each round, the source disk makes a new copy. For other items, the numbers of copies are doubled if possible. Consider Round 4. Since there are four copies of item 1, only two copies need to be created to make $|D_i|/2 = 6$ copies. For items 2 and 3, we can double the number of copies, and a new copy for item 4 is created by the source disk.

Without loss of generality, we assume that $|D_1| \geq |D_2| \geq \cdots \geq |D_\Delta|$ (otherwise renumber the items). Let d_i be the largest index such that $2^{d_i} \leq |D_i|$. For example, if $|D_i| = 12$, then $d_i = 3$.

Phase I. At the t-th round, we do the following.

1. The source disk s creates a new copy for item t if $t \leq \Delta$.

2. For items j ($j < t$), double the number of copies until the number of copies becomes $\lfloor |D_j|/2 \rfloor$.

   In other words, if the current number of copies of item j is less than or equal to $2^{d_j - 2}$,

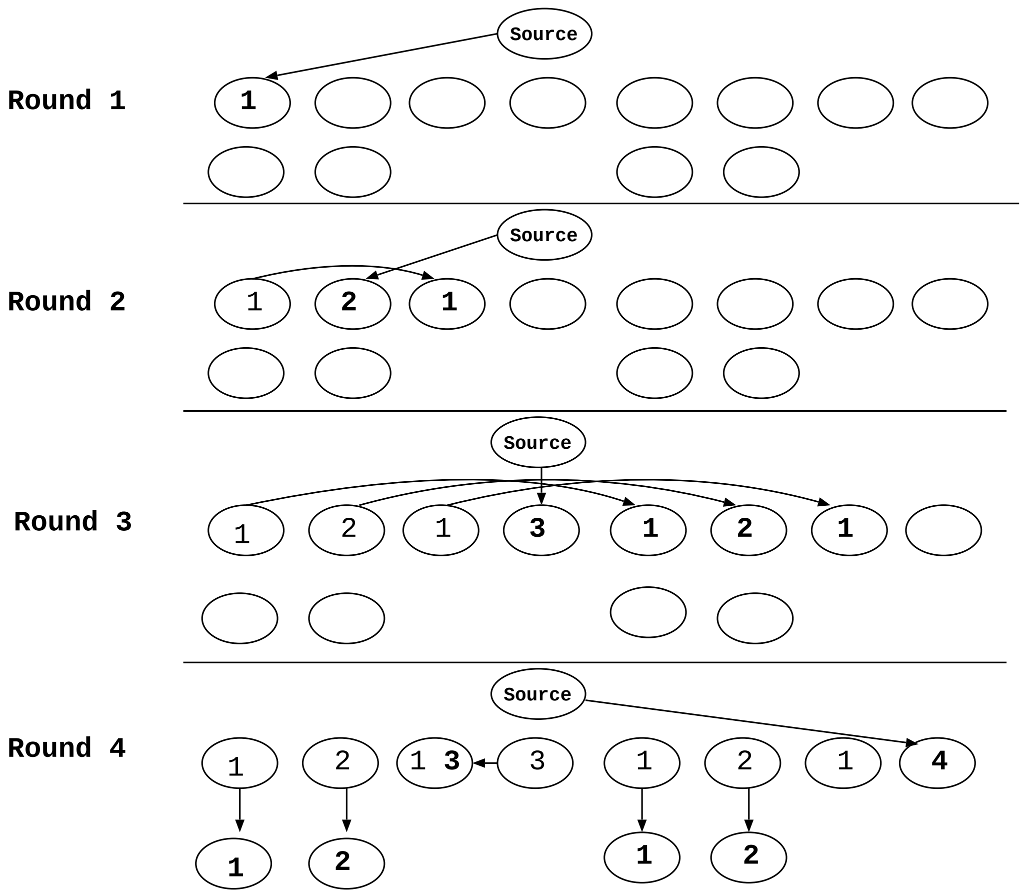


Figure 3.1: An example of a single-source broadcast instance.

every disk having item j makes another copy of it so that the number of copies is doubled. Otherwise if the current number of copies of item j is 2^{d_j-1}, then only $\lfloor |D_j|/2 \rfloor - 2^{d_j-1}$ disks need to make copies (thus the number of copies of item j becomes exactly $\lfloor |D_j|/2 \rfloor$).

Phase II. After Phase I, each item j has exactly $\lfloor |D_j|/2 \rfloor$ copies. Therefore, at each round, we finish the migration of one item. At the t-th round, we copy item t from the current copies to the remaining $\lfloor |D_t|/2 \rfloor$ disks in D_t which did not receive item t as yet, and we use the source disk to make one more copy if $|D_t|$ is odd.

Figure 3.2 and Figure 3.3 show an example of data transfers in Phase I and Phase II, where $|D_1|$, $|D_2|$, and $|D_3|$ are 16, 12, and 8, respectively (i.e., all $|D_i|$ are even).

Since migrations of several items happen at the same time in Phase I, and D_i's are arbitrary, we need to carefully choose which subset of disks will participate in the migration of each item. We explain the details of how we can perform Phase I without conflicts in the next section.

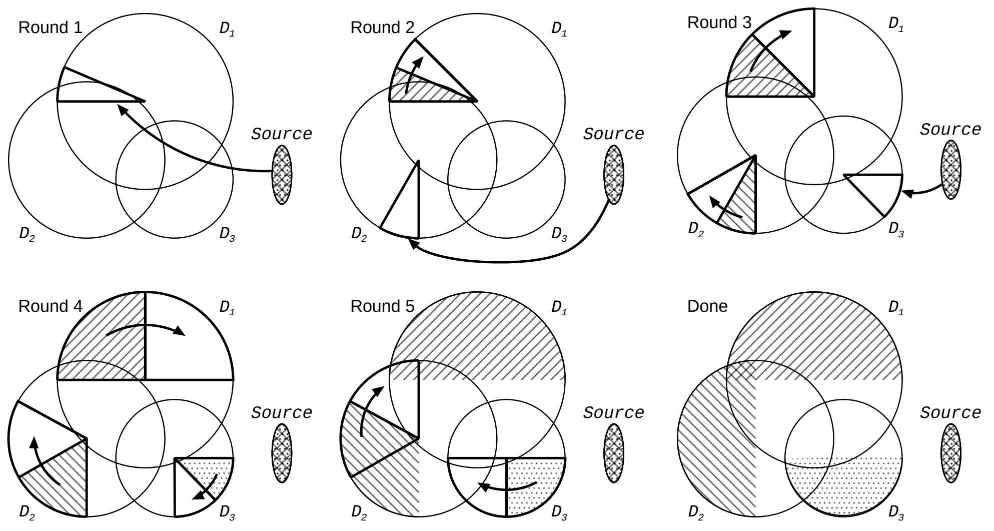


Figure 3.2: An example of Phase I in Algorithm Single-Source Multicast.

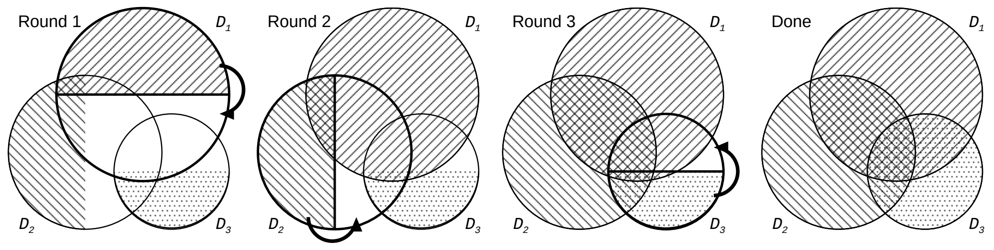


Figure 3.3: An example of Phase II in Algorithm Single-Source Multicast.

3.3 Details of Phase I

Recall that we assume that $|D_1| \geq |D_2| \geq \cdots \geq |D_\Delta|$ and make copies starting from $D_1, D_2, \ldots$. Let D_i^p be the disks in D_i that participate in either sending or receiving item i at the $(i+p)$-th round. Then the size of D_i^p should be

$$|D_i^p| = \begin{cases} 2^p & \text{if } p \leq d_i - 1 \\ 2\left(\left\lfloor \frac{|D_i|}{2} \right\rfloor - 2^{d_i - 1}\right) & \text{if } p = d_i \end{cases}$$

D_i^0 is the first disk receiving i from the source s, and the size is doubled at each round until the number of copies becomes $\lfloor |D_i|/2 \rfloor$. Figure 3.4 shows how disks in D_i^p behave in Phase I where $|D_i| = 2^4 + 2^2 + 2^1$.

We build D_Δ^p first as follows. For $D_\Delta^{d_\Delta}$, choose $2(\lfloor |D_\Delta|/2 \rfloor - 2^{d_\Delta - 1})$ disks arbitrarily from D_Δ. When we choose $D_\Delta^{d_\Delta - 1}$, choose $\lfloor |D_\Delta|/2 \rfloor - 2^{d_\Delta - 1}$ disks from $D_\Delta^{d_\Delta}$ and choose $2^{d_\Delta} - \lfloor |D_\Delta|/2 \rfloor$ disks from $D_\Delta \setminus D_\Delta^{d_\Delta}$ so that the size of $D_\Delta^{d_\Delta - 1}$ is $2^{d_\Delta - 1}$. For each D_Δ^p ($p < d_\Delta - 1$), choose 2^p disks from D_Δ^{p+1}. Note that for all p, exactly half of disks in D_Δ^p are included in D_Δ^{p-1} (which will be used as senders) and the remaining half is not included (which will be used as receivers). For example, if the size of D_Δ is 12, then we choose 1, 2, 4 disks for D_Δ^p ($p = 0, 1, 2$) respectively and for D_Δ^3, include 2 disks from D_Δ^2 and 2 disks from $D_\Delta \setminus D_\Delta^2$.

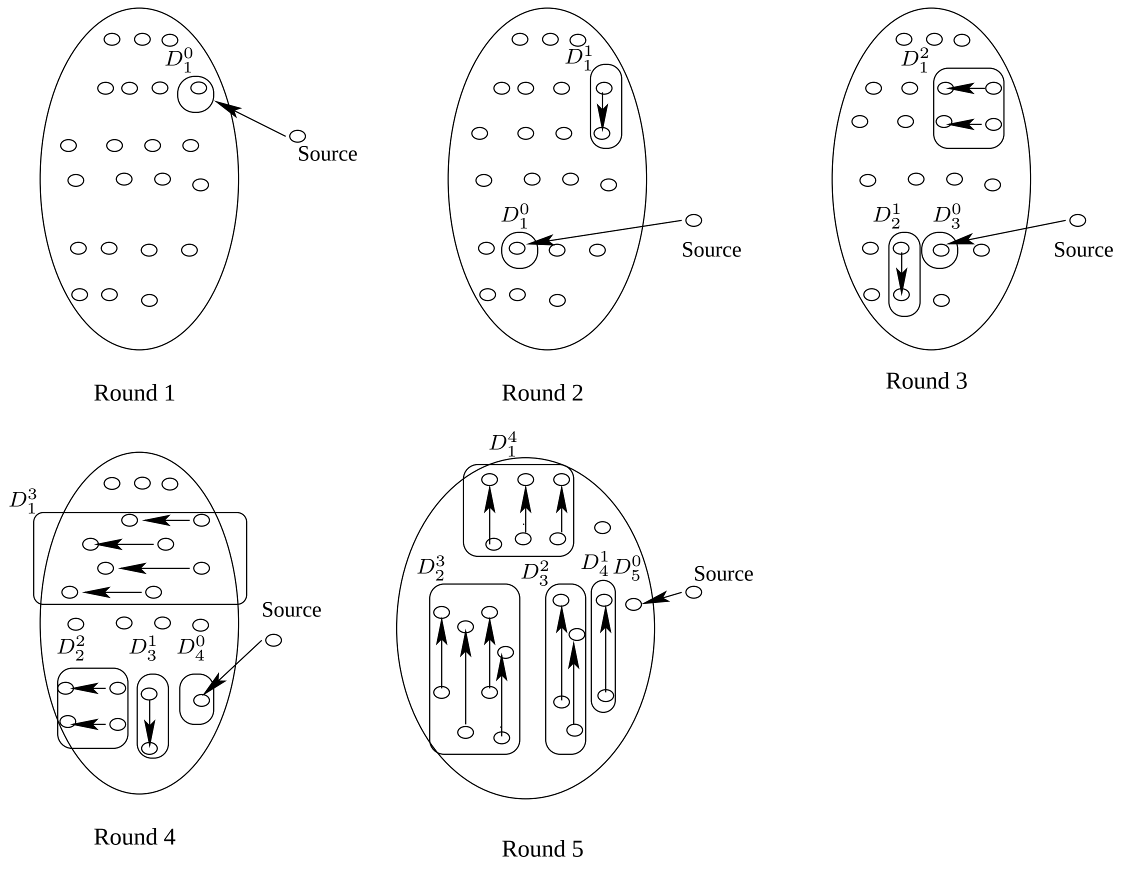


Figure 3.4: Behavior of disks in D_i in Phase I.

We now decide D_i^p, given all $D_j^{p'}$ ($j > i$). At the $(i + p)$-th round (when disks in D_i^p

participate in migration for item i), disks in D_j^{i+p-j} for all j ($i < j \leq \min(i+p, \Delta)$) also participate in either sending or receiving item j at the same time. We have to decide which disks belong to D_i^p to avoid conflicts with D_j^{i+p-j}s ($j > i$).

Consider $D_i^{d_i}$. The set should not be overlapped with any set $D_j^{i+d_i-j}$ ($j > i$) since they also participate in migration at the $(i + d_i)$-th round. Therefore we define

$$D_i' = D_i - \bigcup_{j=i+1}^{\Delta} D_j^{i+d_i-j}$$

and choose $D_i^{d_i}$ from D_i'. Similarly, set $D_i^{d_i-1}$ should not be overlapped with any set $D_j^{i+d_i-1-j}$, $j > i$. Therefore, we define

$$D_i'' = D_i - \bigcup_{j=i+1}^{\Delta} D_j^{i+d_i-1-j}$$

and choose set $D_i^{d_i-1}$ from D_i''. Figure 3.5 shows how to choose disks in D_i^p in Phase I where $|D_i| = 2^4 + 2^2 + 2^1$. Note that half of $D_i^{d_i}$ should be included in $D_i^{d_i-1}$ (to be senders) and the remaining half should be excluded from $D_i^{d_i-1}$ (to be receivers). For D_i^p ($p < d_i - 1$), we can choose half the disks from D_i^{p+1}.

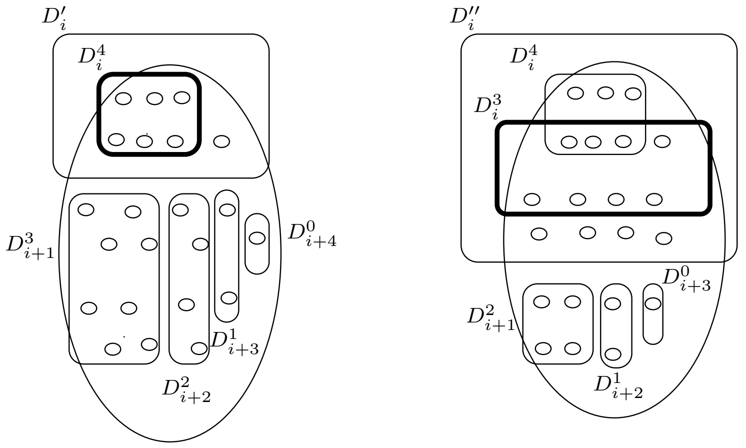


Figure 3.5: Choosing disks in D_i^p in Phase I.

Lemma 3.3.1 *We can find a migration schedule in which we perform every round in phase I without conflicts.*

Proof First we show that there are enough disks to build D_i^p as described above. Because $|D_j^p| \leq 2^p$,

$$\begin{aligned} |D_i''| &= \left| D_i - \bigcup_{j=i+1}^{\Delta} D_j^{i+d_i-1-j} \right| \\ &\geq |D_i| - \sum_{j=i+1}^{\Delta} 2^{i+d_i-1-j} \\ &\geq |D_i| - \sum_{m=0}^{d_i-2} 2^m \\ &> |D_i| - 2^{d_i-1} \end{aligned}$$

Therefore, even after excluding $\lfloor |D_i|/2 \rfloor - 2^{d_i-1}$ disks in D_i' from D_i'', we have at least $|D_i|/2 \geq 2^{d_i-1}$ disks, from which we can take 2^{d_i-1} disks for $D_i^{d_i-1}$. Also we know that

$$|D_i'| = \left| D_i - \bigcup_{j=i+1}^{\Delta} D_j^{i+d_i-j} \right| > |D_i| - 2^{d_i}.$$

Because we only need $2\lfloor |D_i|/2 \rfloor - 2^{d_i}$ disks for $D_i^{d_i}$, we have enough disks to choose from.

Now we argue that there is no conflict in performing the migration if we do the migration according to D_i^p. Since $D_i^{d_i} \subset D_i'$ and $D_i' \bigcap D_j^{i+d_i-j} = \emptyset$ $(j > i)$, there is no conflict between i and j at the $(i + d_i)$-th round. For $p \leq d_i - 1$, since $D_i^p \subset D_i''$ and $D_i'' \bigcap D_j^{i+p-j} = \emptyset$ $(j > i)$, there is no conflict between i and j at the $(i + p)$-th round. Therefore, we can perform the migration in Phase I without conflicts. □

3.4 Analysis

We prove that our algorithm uses at most Δ more rounds than the optimal solution for single-source multicasting. Let us denote the optimal makespan of an migration instance I as $C(I)$.

Theorem 3.4.1 *For any migration instance I, $C(I) \geq \max_{1 \leq i \leq \Delta}(i + \lfloor \log |D_i| \rfloor)$.*

Proof Consider the instance where there is no overlap among D_i's. After a disk in D_i receives i from s for the first time, we need at least $\lfloor \log |D_i| \rfloor$ more rounds to make all disks in D_i receive i even if s copies item i several times after the first copy. Therefore, $C(I) \geq \max_{1 \leq i \leq \Delta}(f(i) + \lfloor \log |D_i| \rfloor)$

where $f(i)$ is the round when D_i receives the first copy from s. Because s can be involved in copying only one item at a time, $f(i) \neq f(j)$ if $i \neq j$. Also copying the same item from s more than once during the first Δ rounds will only increase $f(i)$ of some sets. Therefore, $C(I)$ can be minimized by choosing $f(i)$ as a permutation of $1, \ldots, \Delta$. Now we show that $\max_{1 \leq i \leq \Delta}(f(i) + \lfloor \log |D_i| \rfloor) \geq \max_{1 \leq i \leq \Delta}(i + \lfloor \log |D_i| \rfloor)$ for any permutation $f(i)$. Suppose there is a set D_i that $f(i) \neq i$ when $\max_{1 \leq i \leq \Delta}(f(i) + \lfloor \log |D_i| \rfloor)$ is minimum. Let D_i be the set which have the smallest $f(i)$ among such sets. Then $f(i) < i$ and there should be a D_j such that $j = f(i)$ and $f(j) > j$. Even if we exchange the order of two sets, the value does not increase because

$$\begin{aligned} \max(f(i) + \lfloor \log |D_i| \rfloor, f(j) + \lfloor \log |D_j| \rfloor) &= f(j) + \lfloor \log |D_j| \rfloor \\ &\geq \max(j + \lfloor \log |D_j| \rfloor, f(j) + \lfloor \log |D_i| \rfloor). \end{aligned}$$

Thus when $f(i) = i$ for all i, $\max_{1 \leq i \leq \Delta}(f(i) + \lfloor \log |D_i| \rfloor)$ is minimized. □

Lemma 3.4.2 *The total makespan of our algorithm is at most* $\max_{1 \leq i \leq \Delta}(i + \lfloor \log |D_i| \rfloor) + \Delta$.

Proof In phase I, D_i receives i from s at the i-th round for the first time. Because the number of copies doubles until it reaches $\lfloor |D_i|/2 \rfloor$, the number of copies of item i reaches $\lfloor |D_i|/2 \rfloor$ in $i + \lfloor \log |D_i| \rfloor$ rounds. Phase II takes at most Δ rounds because we finish one item in each round. Therefore, the lemma follows. □

Corollary 3.4.3 *The total makespan of our algorithm is at most the optimal makespan plus* Δ.

Proof Follows from Lemma 3.4.1 and Lemma 3.4.2. □

Theorem 3.4.4 *We have a 2-approximation algorithm for the single-source multicasting problem.*

Proof Because $\Delta \leq \max_{1 \leq i \leq \Delta}(i + \lfloor \log |D_i| \rfloor)$, the algorithm is 2-approximation. □

Chapter 4

Multi-Source Broadcasting

Suppose we have N disks and Δ data items. Disk i, $1 \leq i \leq \Delta$, has item numbered i. The goal for *multi-source broadcasting problem* is to send each item i to all N disks, for all i. In this chapter[1], we consider this problem and present an algorithm that takes at most 3 more rounds than the optimal solution.

4.1 Problem Specification

Suppose we have N disks and Δ data items. The multi-source broadcast problem is defined as follows:

Multi-source broadcast. There are Δ data items, each stored separately at a single disk. These need to be broadcast to all disks. We assume that data item i is stored on disk i, for $i = 1 \ldots \Delta$. Our goal is to find a schedule using the minimum number of rounds, that is, minimizing the makespan, subject to the same communication model described in Section 3.1.1.

4.2 Algorithm Multi-Source Broadcast

For the high-level description we assume for simplicity that N is a multiple of Δ. The main idea behind the algorithm is the following. We first partition the disks into Δ equally sized groups $G_1, \ldots, G_\Delta$. Group G_i contains the source disk of item i. We now perform broadcasts in parallel for each group so that each disk in group G_i contains item i. We now make $\frac{N}{\Delta}$ new groups of size Δ by picking one disk from each group. Since each disk contains a different item, this is exactly the gossip problem for a set of Δ disks. We solve all these gossip problems in parallel. Now all disks contain all the items. Figure 4.1 shows an example to illustrate this main idea, when $N = 18$ and $\Delta = 6$. The actual algorithm is a little more complicated since it works for arbitrary values

[1]This is joint work with S. Khuller and Y. Kim [61, 58].

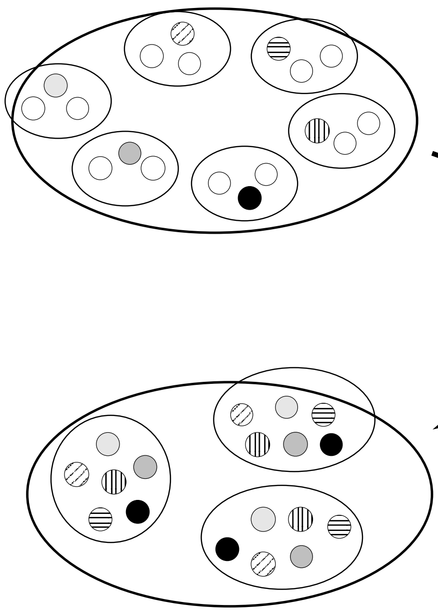


Figure 4.1: An example to illustrate the main idea behind Algorithm Multi-Source Broadcast.

of N, and is described as follows.

1. We partition the N disks into Δ sets G_i such that disk $i \in G_i$, for all $i = 1 \ldots \Delta$. Let q be $\lfloor \frac{N}{\Delta} \rfloor$ and r be $N - q\Delta$. $|G_i| = q + 1$ for $i = 1 \ldots r$, and $|G_i| = q$ for $i = r + 1 \ldots \Delta$. We do broadcasts in parallel for each group G_i of item i. This takes $\max\lceil \log |G_i| \rceil$ rounds by doubling the number of items in each round. (Each disk that receives an item, sends it out in each subsequent round until all the disks in the group have the item.)

2. We now partition the N disks into $q-1$ groups of size Δ each, by picking one disk from each G_i, and one group of size $\Delta + r$ that contains all the remaining disks.

3. Consider the first $q-1$ groups; each group consists of Δ disks, with each having a distinct item. Using the gossiping algorithm in [11], every disk in the first $q-1$ groups receives all Δ items in 2Δ rounds[2].

4. In parallel with Step 3, consider the last gossiping group, for each of the items numbered

[2]The number of rounds required is 2Δ if Δ is odd, otherwise it is $2(\Delta - 1)$

$1, \ldots, r$, there are exactly two disks having this item. For each of the items numbered $r+1, \ldots, \Delta$, there is exactly one disk having this item. Note that each disk has exactly one item. If r is zero, we can finish all transfers in 2Δ rounds using the algorithm in [11]. For non-zero r, we claim that all disks in this gossiping group still receive all items in 2Δ rounds.

We divide the disks in the last gossiping group into two groups, G_X and G_Y of size $\Delta - \lfloor \frac{\Delta - r}{2} \rfloor$ and $r + \lfloor \frac{\Delta - r}{2} \rfloor$ respectively. Figure 4.2 shows how the last group of disks in Step 4 in Algorithm Multi-Source Broadcast is partitioned into G_X and G_Y. Each of the items numbered $1, \ldots, r$ appear in both G_X and G_Y; disks having items $r+1, \ldots, \Delta - \lfloor \frac{\Delta - r}{2} \rfloor$ are in G_X, and the remaining disks (having items $\Delta - \lfloor \frac{\Delta - r}{2} \rfloor + 1, \ldots, \Delta$) are in G_Y. Note that the sizes of the two groups differ by at most 1. The general idea of the algorithm is as follows (The details of these step are non-trivial and covered in the proof of Lemma 4.3.1):

(a) The Gossip algorithm in [11] is applied to each group in parallel. After this step, each disk has all the items belonging to its group.

(b) In each round, disks in G_Y send item i to disks in G_X, where i is $\Delta - \lfloor \frac{\Delta - r}{2} \rfloor + 1, \ldots, \Delta$. Note that only disks in G_Y have these items, but not the disks in G_X. Since the group sizes differ by at most 1, the number of rounds required is about the same as the number of items transferred.

(c) The step is similar to the above step but in the reverse direction. Item i, where i is $r+1, \ldots, \Delta - \lfloor \frac{\Delta - r}{2} \rfloor$, is sent from G_X to G_Y.

Thus, our algorithm takes $\lceil \log \frac{N}{\Delta} \rceil + 2\Delta$ rounds. The first term comes from the broadcast in Step 1. The second term comes from the number of rounds required by the gossiping algorithm in Steps 3 and 4.

4.3 Analysis

Lemma 4.3.1 *For a group of disks of size $\Delta + r$, where $1 \leq r < \Delta$, if every disk has one item, exactly two disks have items $1, \ldots r$, and exactly one disk has item $r+1, \ldots, \Delta$, all disks can receive*

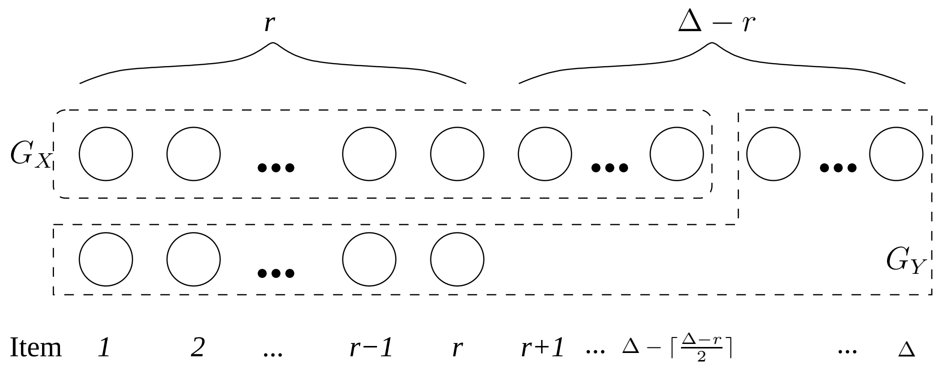


Figure 4.2: Partitioning the last group of disks in Step 4 in Algorithm Multi-Source Broadcast into G_X and G_Y.

all Δ *items in* 2Δ *rounds.*

Proof We have three cases.

Case I: If $\Delta + r$ is even: Step 4a can be done in $2(\Delta - \frac{\Delta-r}{2})$ rounds because $\Delta - \frac{\Delta-r}{2}$ is the group size. In Steps 4b and 4c, we can finish one item in one round since the size of the two groups is the same. All disks can participate in transferring data without any conflict. There are $(\frac{\Delta-r}{2}) + (\Delta - r - \frac{\Delta-r}{2})$ items to be sent in these 2 steps. Thus, the total number of rounds needed is $(2(\Delta - \frac{\Delta-r}{2})) + (\frac{\Delta-r}{2}) + (\Delta - r - \frac{\Delta-r}{2}) = 2\Delta$.

Case II: If $\Delta + r$ is odd and $|G_X| = \Delta - \frac{\Delta-r-1}{2}$ is even: Step 4a can be done in $2(\Delta - \frac{\Delta-r-1}{2} - 1)$ rounds. In Step 4b, $\frac{\Delta-r-1}{2}$ items have to be copied to G_X but $|G_Y|$ is smaller than $|G_X|$ by one. Instead of keeping one disk idle all the time, we shift the disk not receiving an item in each round. After this step finishes, only $\frac{\Delta-r-1}{2}$ disks in G_X miss an item, while other disks in G_X receive all $\frac{\Delta-r-1}{2}$ items. By using one more round, all disks in G_X can receive all items needed from G_Y. In Step 4c, $\Delta - r - \frac{\Delta-r-1}{2}$ items have to be copied to G_Y, and we have enough source disks in G_X. Thus, it requires $(2(\Delta - \frac{\Delta-r-1}{2} - 1)) + (\frac{\Delta-r-1}{2} + 1) + (\Delta - r - \frac{\Delta-r-1}{2}) = 2\Delta$ rounds.

Case III: If $\Delta + r$ is odd and $|G_X| = \Delta - \frac{\Delta-r-1}{2}$ is odd: Since $|G_X|$ is odd, Step 4a takes $2(\Delta - \frac{\Delta-r-1}{2})$ rounds. We claim that in this step, in addition to receiving items from its group, all disks in G_X, except the disk that has item 1 originally, have item Δ, and all disks in G_Y have item $\Delta - \frac{\Delta-r-1}{2}$ (i.e., the highest-numbered item in G_X). We use the algorithm in [11] to form a schedule for G_X with the constraint that (i) the disk that has item 1 originally should be idle during the first two rounds, and (ii) the disk that received item $\Delta - \frac{\Delta-r-1}{2}$, except the disk having item 1 originally, should be idle in the next two rounds. It is not difficult to check that such a schedule exists. The disk that has item $\Delta - \frac{\Delta-r-1}{2}$ originally is idle during the last 2 rounds. We sort the disks in G_X according to the item number it has, and label the disks as disk $1, 2, \ldots, \Delta - \frac{\Delta-r-1}{2}$. We also sort disks in G_Y, but label the disks as $2, 3, \ldots, \Delta - \frac{\Delta-r-1}{2}$. Disk 1 in G_Y is an imaginary disk which does not exist. Whenever disk x and y in G_X exchange data in the gossiping schedule of G_X, disk x and y in G_Y also exchange data in the same round. Moreover, starting at round 3, the idle disk in G_X, which should have item $\Delta - \frac{\Delta-r-1}{2}$, will exchange data with the idle disk in G_Y, which should have item Δ. If a disk in G_Y is supposed to exchange data with disk 1 in G_Y (i.e., the imaginary disk), the disk would actually be idle in that round. Here we give an example, as shown in Figure 4.3, with $\Delta = 6$ and $r = 3$. G_X has 5 items and G_Y has 4 items. Performing a gossip operation in G_X and G_Y takes 10 rounds (twice the size of the larger group). Notice that in 10 rounds, we are also able to send the highest-numbered item (item 6) in G_Y to all the disks in G_X except disk 1. At the same time, we are able to send the highest-numbered item (item 5) in G_X to all the disks in G_Y. Thus when we send items between G_X and G_Y we are able to save rounds.

Note that we just exploit the idle cycles in the gossiping schedule. The number of rounds required is still $2(\Delta - \frac{\Delta-r-1}{2})$. One disk in G_X always exchanges data with one disk in G_Y except in the first two rounds. All disks in G_X and G_Y, except disk 1 in G_X, receive one extra item from the other group.

In Steps 4b and 4c, the analysis is similar to that in **Case II** except that we save one round in each step because each disk has already received one item from the other group in Step 4a. The disk in G_X which does not have item Δ, can receive it in the last round of Step 4b because $\frac{\Delta-r-1}{2} + 1 \leq |G_Y|$.

Thus, the total number of rounds is $2(\Delta - \frac{\Delta-r-1}{2}) + (\frac{\Delta-r-1}{2}) + (\Delta - r - \frac{\Delta-r-1}{2} - 1) = 2\Delta$. $\square$

The proof of the following theorem follows trivially from the above Lemma.

Theorem 4.3.2 *Our multi-source broadcast algorithm takes* $\lceil \log \frac{N}{\Delta} \rceil + 2\Delta$ *rounds.*

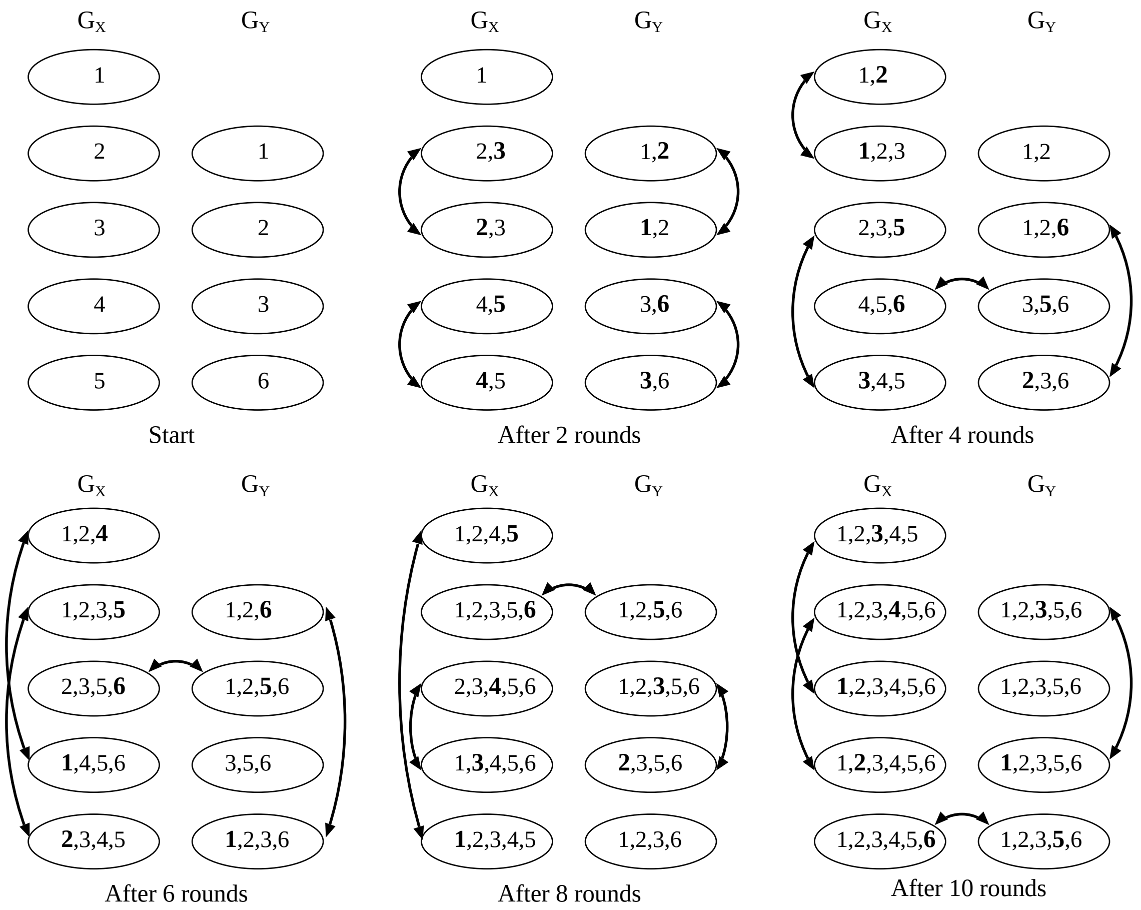


Figure 4.3: An example of Case III in Algorithm Multi-Source Broadcasting with $\Delta = 6$ and $r = 3$. Recently received items are in bold.

To show our algorithm is close to optimal, we will show a lower bound of any algorithm for the problem.

Theorem 4.3.3 *The time required for any migration instance of multi-source broadcasting is at least* $\lfloor \log \frac{N}{\Delta} \rfloor + 2(\Delta - 1)$.

Proof Consider a transfer graph of the optimal solution, where vertices are disks and an edge from i to j represents one item that is copied from disk i to disk j at a certain time. Each of the

Δ source disks needs $\Delta - 1$ items. For each of the remaining $N - \Delta$ disks, they need all Δ items. Therefore, there should be $\Delta(\Delta-1)+(N-\Delta)\Delta = \Delta(N-1)$ edges (corresponding to the number of transfers).

In the initial $\lfloor \log \frac{N}{\Delta} \rfloor$ rounds, some disks have to be idle because of the limited number of sources. For example, if there are x non-empty disks at a certain round, one can perform at most x transfers. If all the transfers send data to other empty disks, one can perform $2x$ transfers in the next round, while other schemes cannot support $2x$ transfers in the next round. Therefore, the best scheme is to keep on doubling all items in each round until all disks have at least one item. This takes at least $\lfloor \log \frac{N}{\Delta} \rfloor$ rounds. Now, at most $N - \Delta$ transfers are done.

The total degree of the transfer graph after removing the edges corresponding to the first $\lfloor \log \frac{N}{\Delta} \rfloor$ rounds is at least $2(\Delta(N-1) - (N-\Delta)) = 2N(\Delta-1)$. Note that each disk can send or receive only one item in a round. All N disks can reduce the degrees of the graph by N in a round. The total time is at least $\lfloor \log \frac{N}{\Delta} \rfloor + \frac{2N(\Delta-1)}{N} = \lfloor \log \frac{N}{\Delta} \rfloor + 2(\Delta-1)$. $\square$

Thus, our solution takes no more than 3 rounds more as compared to the optimal.

Chapter 5

Multi-Source Multicasting

Suppose we have N disks. Disk i, $1 \leq i \leq \Delta \leq N$, has data item i. The goal for *multi-source multicasting problem* is to send item i to a subset D_i of disks that do not have item i. (Hence $i \notin D_i$.) In this chapter[1], we first present a simple polynomial-time 4-approximation algorithm for this problem. We then show how to improve it to give a $(3 + o(1))$-approximation algorithm. We also present a 3-approximation algorithm for the special case in which the source disks are not in any subset D_i. After that we present a 3-approximation algorithm that allows the use of bypass disks, where bypass disks are disks that are used as temporary holding points of data. We also look at a communication model where the network does not have unlimited bandwidth. Under this model, we have constant factor approximation algorithms for the single-source multicast, multi-source broadcast, and multi-source multicast problems. Lastly in Section 5.6 we show that finding a schedule with minimum number of rounds is NP-hard.

5.1 Problem Specification

Suppose we have N disks and Δ data items. The multi-source multicast problem is defined as follows:

Multi-source multicast. There are Δ data items, each stored separately at a single disk. Data item i needs to be sent to a specified subset D_i of disks. We assume that data item i is stored on disk i, for $i = 1 \ldots \Delta$. Our goal is to find a schedule using the minimum number of rounds, that is, minimizing the makespan, subject to the same communication model described in Section 3.1.1.

[1]This is joint work with S. Khuller and Y. Kim [61, 58].

5.1.1 Background

Our algorithms make use of a known result on edge coloring of multi-graphs. Given a graph G with max degree Δ_G and multiplicity μ the following result is known (see [14] for example). Let χ' be the edge chromatic number of G.

Theorem 5.1.1 *(Vizing [90]) If G has no self-loops then $\chi' \leq \Delta_G + \mu$.*

5.2 Algorithm Multi-Source Multicast

A high-level description of the algorithm is as follows: we first create a small number of copies of each data item i (the exact number of copies will be dependent on $|D_i|$). We then assign each newly created copy to a set of disks in D_i, such that it will be responsible for providing item i to those disks. The assignment will be used to construct a transfer graph, where each directed edge labeled i from v to w indicates that disk v must send item i to disk w. We will then use an edge-coloring of this graph to obtain a valid schedule [14]. The main difficulty here is that a disk containing an item as its source, may also be the destination for several other data items. Before we describe the algorithm, we first define a value β which is used in the algorithm.

We define β to be $\max_{j=1\ldots N} |\{i | j \in D_i\}|$. In other words, β is an upper bound on the number of different sets D_i to which a disk j may belong. Note that β is a lower bound on the optimal number of rounds, since the disk that attains the maximum, needs at least β rounds to receive all the items i such that $j \in D_i$, because it can receive at most one item in each round.

Algorithm Multi-Source Multicast

1. We first compute a disjoint collection of subsets of disks $G_i, i = 1 \ldots \Delta$. We ensure that $G_i \subseteq D_i$ and $|G_i| = \left\lfloor \frac{|D_i|}{\beta} \right\rfloor$. (In Lemma 5.2.1, we will show how such G_i's can be obtained by using network flows.)

2. Since the G_i's are disjoint, we have the source for item i (namely disk i) send the data to the set G_i using $\lceil \log |D_i| \rceil + 1$ rounds as shown in Lemma 5.2.2. Note that disk i may itself belong to some set G_j. Let $G'_i = \{i\} \cup G_i$. In other words, G'_i is the set of disks that have

item i at the end of this step.

3. We now create a transfer graph as follows. Each disk is a node in the graph. We add directed edges from each disk in G'_i to disks in $D_i \setminus G_i$ such that each node in G'_i sends item i to at most $\beta - 1$ disks and each node in $D_i \setminus G_i$ receives item i from one disk in G'_i. (In Lemma 5.2.3 we show how that this can be done.) This ensures that each disk in D_i receives item i, and that each disk in G'_i does not send item i to more than $\beta - 1$ disks.

4. We now find an edge coloring of the transfer graph (which is actually a multigraph) and the number of colors used is an upper bound on the number of rounds required to ensure that each disk in D_j gets item j. (In Lemma 5.2.4 we derive an upper bound on the degree of each vertex in this graph.)

5.2.1 Analysis

Lemma 5.2.1 *(Step 1) There is a way to choose disjoint sets G_i for each $i = 1 \ldots \Delta$, such that $|G_i| = \left\lfloor \frac{|D_i|}{\beta} \right\rfloor$ and $G_i \subseteq D_i$.*

Proof First note that the total size of the sets G_i is at most N.

$$\sum_i |G_i| \leq \sum_i \frac{|D_i|}{\beta} = \frac{1}{\beta} \sum_i |D_i|.$$

Note that $\sum_i |D_i|$ is at most βN by definition of β. This proves the upper bound of N on the total size of all the sets G_i.

We now show how to find the sets G_i. We create a flow network with a source s and a sink t. In addition we have two sets of vertices U and W. The first set U has Δ nodes, each corresponding to a disk that is the source of an item. The set W has N nodes, each corresponding to a disk in the system. We add directed edges from s to each node in U, such that the edge (s, i) has capacity $\left\lfloor \frac{|D_i|}{\beta} \right\rfloor$. We also add directed edges with infinite capacity from node $i \in U$ to $j \in W$ if $j \in D_i$. We add unit capacity edges from nodes in W to t. We find a max-flow from s to t in this network. The min-cut in this network is obtained by simply selecting the outgoing edges from s. We can find a fractional flow of this value as follows: saturate all the outgoing edges from

s. From each node i there are $|D_i|$ edges to nodes in W. Suppose $\lambda_i = \left\lfloor \frac{|D_i|}{\beta} \right\rfloor$. Send $\frac{1}{\beta}$ units of flow along $\lambda_i \beta$ outgoing edges from i. Note that since $\lambda_i \beta \leq |D_i|$ this can be done. Observe that the total incoming flow to a vertex in W is at most 1 since there are at most β incoming edges, each carrying at most $\frac{1}{\beta}$ units of flow. An integral max flow in this network will correspond to $|G_i|$ units of flow going from s to i, and from i to a subset of vertices in D_i before reaching t. The vertices to which i has non-zero flow will form the set G_i. $\square$

Lemma 5.2.2 *Step 2 can be done in* $\max_i \lceil \log |D_i| \rceil + 1$ *rounds.*

Proof First we assume that $\max_i |D_i| > 2$ and $\beta \geq 2$ since otherwise the problem becomes trivial.

We arbitrarily choose a new source disk s'_i in each G_i and send item i from disk i to s'_i. Because a disk i may send item i to s'_i and receive item j if $i = s'_j$, this initial transfer can take 2 rounds unless the transfer makes odd cycles (we will consider the case of odd cycles later).

Because the sets G_i are disjoint, it takes $\lceil \log |G_i| \rceil$ rounds to send item i from s'_i to all disks in G_i. The result follows from considering the non-trivial case where $\beta \geq 2$, $\lceil \log |G_i| \rceil \leq \left\lceil \log \frac{|D_i|}{\beta} \right\rceil \leq \lceil \log |D_i| - 1 \rceil$.

Now let us consider the case of odd cycles. If any G_i in the odd cycle is of size at least 2, then we can break the cycle by selecting other disk in G_i as s'_i. Otherwise if the size of all G_i's is one, then this step can be done in 3 rounds (no broadcasting is needed inside G_i) and therefore the lemma is true. $\square$

Lemma 5.2.3 *Consider a transfer graph only for item i in Step 3. We can construct a transfer graph for item i such that the in-degree of each node in* $D_i \setminus G_i$ *from* G_i *is 1 and the out-degree of each node in* G_i*, to* $D_i \setminus G_i$ *is at most* $\beta - 1$.

Proof We divide each $D_i \setminus G_i$ into disjoint sets $D_{i1}, \ldots, D_{im_i}$ where $m_i = \left\lceil \frac{|D_i|}{\beta} \right\rceil$ such that $|D_{ij}| = \beta - 1$ for $j = 1, \ldots, m_i - 1$ and $|D_{im_i}| = |D_i \setminus G_i| - (\beta - 1)(m_i - 1)$. For each set D_{ij}, we choose a different disk from G'_i and add a directed edge from the disk to all disks in D_{ij}. Because

$|D_{ij}| < \beta$ and each disk in $D_i \setminus G_i$ will have an incoming edge from one disk in G'_i, we have a transfer graph as described in Step 3. □

Lemma 5.2.4 *The in-degree of any disk in the transfer graph is at most β. The out-degree of any disk in the transfer graph is at most $2\beta - 2$. Moreover, the multiplicity of the graph is at most 4.*

Proof Note that each disk i may belong to at most β sets D_j. Due to its membership in set D_j it may have one incoming edge from some disk in G'_j.

The out-degree of disk i is $\beta - 1$ due to membership in the set G'_i. These are the $\beta - 1$ edges added in Step 3. In addition, i may be in some set G_k (and thus in G'_k); this may cause an extra out-degree of $\beta - 1$. This gives a total out-degree of at most $2\beta - 2$.

Each disk can be a source for two items because it can be the original source of an item i and also belongs to G_k $(k \neq i)$. Since the subgraph with edges for only one item is a simple graph, for any pair of disks p, q, there can be two edges from p to q and two more edges in another direction. Therefore, the multiplicity of the transfer graph is at most 4. □

Theorem 5.2.5 *The total number of rounds required for the multi-source multicast is* $\max_i \lceil \log |D_i| \rceil + 3\beta + 3$.

Proof Because of Lemma 5.2.4, we can find an edge coloring of the graph using at most $3\beta + 2$ colors (see Theorem 5.1.1). Combining with Lemma 5.2.2, we can finish the multi-source multicast in $\max_i \lceil \log |D_i| \rceil + 3\beta + 3$ rounds. □

Theorem 5.2.6 *The total number of rounds required for the multi-source multicast problem is at most $4OPT + 2$.*

Proof Let β_j be $|\{i | j \in D_i\}|$, i.e., the number of different sets D_i, that disk j belongs to. Thus, the in-degree of disk j in any solution (not using bypass disks) is β_j. Consider any source disk s_i for item i. In the transfer graph described in Step 3, its total degree is therefore $\beta_{s_i} + (\beta - 1) + (\beta - 1)$. In the optimal solution, the out-degree of any disk s_i must be at least one, since s_i must send its item to some other disk. Thus, $OPT \geq \max_i(\beta_{s_i} + 1)$. The maximum degree of any source disk s_i in the transfer graph is $\max_i \beta_{s_i} + (\beta - 1) + (\beta - 1) \leq OPT + 2\beta - 3$. Consider any disk j which is not the source, its total degree is $\beta_j + (\beta - 1)$. Note that $OPT \geq \max_j \beta_j$ and $\beta \geq 2$, the maximum degree of any non-source disk is $\max_{j \neq s_i} \beta_j + (\beta - 1) = OPT + (\beta - 1) \leq OPT + 2\beta - 3$. Therefore, the maximum degree of the transfer graph is at most $OPT + 2\beta - 3$. We have an algorithm that takes at most $(\max_i \lceil \log |D_i| \rceil + 1) + (OPT + 2\beta - 3) + 4$ rounds. As $\max_i \lceil \log |D_i| \rceil$ and β are also the lower bounds on the optimal number of rounds, the total number of rounds required is at most $4OPT + 2$. □

For the special case in which the source disks are not in any subset D_i, we can develop better bounds.

Corollary 5.2.7 *When the source disks are not in any subset D_i, the total number of rounds required for the multi-source multicast is* $\max_i \lceil \log |D_i| \rceil + 2\beta + 1$.

Proof Step 2 can be done in $\max_i \lceil \log |D_i| \rceil$ rounds since we can save one round to send item i to s'_i. Also as the original sources do not belong to any G_i, the transfer graph in Step 4 has out-degree at most $\beta - 1$ and multiplicity at most 2. Therefore, the corollary follows. □

Thus we have a 3-approximation for this special case.

5.3 $3 + o(1)$-approximation Algorithm

In this section we present a polynomial-time approximation algorithm with a factor of $3 + o(1)$, as β goes to infinity, for the Multi-Source Multicast problem.

In the previous algorithm, the sets G_i were disjoint. When the size of D_i is small, say $2\beta - 1$, the size of G_i is 1, and the sole disk in G_i is responsible for sending data to $\beta - 1$ disks, while disk i is responsible for sending data to the remaining $\beta - 1$ disks. By allowing a disk to belong to multiple G_i sets, we can decrease the number of disks for which disk i is responsible for sending items. The out-degree of a disk in the transfer graph is reduced, and we can obtain a better bound.

Suppose a disk can now belong to upto p ($\leq \beta$) different G_i sets. In other words, imagine that there are p slots in each disk, and each G_i will occupy exactly $\left\lfloor p\frac{|D_i|}{\beta}\right\rfloor$ slots. If G_i occupies a slot in a disk, the disk will be responsible for sending the item to either $\left\lfloor \frac{\beta}{p}\right\rfloor - 1$ or $\left\lceil \frac{\beta}{p}\right\rceil - 1$ disks in $D_i \setminus G_i$.

Changes to the algorithm

- In Step 1, we create a modified flow network to compute a (not necessarily disjoint) collection of subsets G_i, where $|G_i|$ is $\left\lfloor p\frac{|D_i|}{\beta}\right\rfloor$. In addition, each disk belongs to at most p subsets. We show in Lemma 5.3.1 how such G_i's can be obtained.

- In Step 2, although the G_i's are not disjoint, sending items from s_i to G_i is actually another smaller multi-source multicast problem, where β', the upper bound on the number of different destination sets (G_i) to which a disk j in some G_i may belong, is p. Lemma 5.3.2 describes the details.

- In Step 3, if G_i occupies a slot in disk j, we would like the disk to satisfy either $\left\lfloor \frac{\beta}{p}\right\rfloor - 1$ or $\left\lceil \frac{\beta}{p}\right\rceil - 1$ disks in $D_i \setminus G_i$. Moreover, we would like to keep the total out-degree of disk j to be at most $\beta - p$, while disks in G_i together have to satisfy $\left\lfloor \left\lfloor p\frac{|D_i|}{\beta}\right\rfloor (\frac{\beta}{p} - 1)\right\rfloor$ disks in $D_i \setminus G_i$. We show in Lemma 5.3.3 how this can be achieved by a network flow computation. We also show the source s_i is responsible for at most $\left\lceil \frac{\beta}{p}\right\rceil$ disks.

Lemma 5.3.1 *In Step 1, there is a way to choose sets G_i for each $i = 1 \ldots \Delta$, such that G_i occupies exactly one slot in each of $\left\lfloor p\frac{|D_i|}{\beta}\right\rfloor$ disks, and $G_i \subseteq D_i$. Moreover, each disk has p slots.*

Proof

The basic idea of the proof is similar to that of Lemma 5.2.1.

First note that we have enough slots for G_i (we have N disks and each disk has p slots).

$$\sum_i \left\lfloor p\frac{|D_i|}{\beta}\right\rfloor \leq \frac{p}{\beta}\sum_i |D_i| \leq \frac{p}{\beta}(\beta N) = pN.$$

Now we show how to assign G_i to the slots using a flow network. We create a flow network with a source s and a sink t. We also have two sets of vertices U and W. The first set U has Δ nodes, each corresponding to an item. The set W has N nodes, each corresponding to a disk. We add directed edges from s to each node i in U with capacity $\lambda_i = \left\lfloor p\frac{|D_i|}{\beta} \right\rfloor$. We add unit capacity edges from node $i \in U$ to $j \in W$ if $j \in D_i$. We also add edges with capacity p from nodes in W to t. We find a max-flow from s to t in this network. We can find a fractional flow of this value as follows: saturate all the outgoing edges from s. From each node i there are $|D_i|$ edges to nodes in W. Send $\lambda_i \frac{1}{|D_i|}$ units of flow along each of the $|D_i|$ outgoing edges from i. Note that since $\lambda_i \frac{1}{|D_i|} \leq \frac{p}{\beta} \leq 1$ this can be done. Observe that the total incoming flow to a vertex in W is at most p since there are at most β incoming edges, each carrying at most $\lambda_i \frac{1}{|D_i|} \leq \frac{p}{\beta}$ units of flow. The min-cut in this network is obtained by simply selecting the outgoing edges from s. An integral max flow in this network will correspond to $|G_i|$ units of flow going from s to i, and from i to a subset of vertices in D_i before reaching t. The vertices to which i has non-zero flow will form the set G_i. The unit capacity edges between U and W ensures that G_i only occupies one slot in each disk, and thus $|G_i|$ is exactly $\left\lfloor p\frac{|D_i|}{\beta} \right\rfloor$. □

Lemma 5.3.2 *Step 2 can be done in* $\max_i \log \left\lfloor p\frac{|D_i|}{\beta} \right\rfloor + 3p + 4$ *steps.*

Proof

Observe that sending items from disk i to G_i is just another smaller multi-source multicast problem. The upper bound on the number of different destination sets (G_i) to which a disk j in some G_i may belong is p. Therefore, using the 4-approximation algorithm described in the previous section, we can send items to all disks in G_i in $(\max_i \log \left\lfloor p\frac{|D_i|}{\beta} \right\rfloor + 2) + (p + ((p-1) + (p-1))) + 4 = \max_i \log \left\lfloor p\frac{|D_i|}{\beta} \right\rfloor + 3p + 4$ rounds (by Theorem 5.2.5). □

Lemma 5.3.3 *In Step 3, we can find a transfer graph to satisfy all requests in* $D_i \setminus G_i$*, where the in-degree is at most* β*, the out-degree is at most* $(\beta - p) + \left\lceil \frac{\beta}{p} \right\rceil$*, and the multiplicity is at most* $2(p+1)$*.*

Proof To find out how many disks (in $D_i \setminus G_i$) a disk j in G_i should send item i to, while satisfying the constraints stated in the description of **Changes to the algorithm**, we create a flow network with a source s and a sink t. We also have two sets of vertices U and W. The first set U has Δ nodes, each corresponding to an item. The set W has N nodes, each corresponding to a disk. We add directed edges from s to each node i in U with capacity $\gamma_i = \left\lfloor \left\lfloor p\frac{|D_i|}{\beta} \right\rfloor (\frac{\beta}{p} - 1) \right\rfloor$. We add edges from node $i \in U$ to $j \in W$ if $j \in G_i$ with capacity $\left\lceil \frac{\beta}{p} \right\rceil - 1$. We also add edges with capacity $\beta - p$ from nodes in W to t. We find a max-flow from s to t in this network. The min-cut in this network is obtained by simply selecting the outgoing edges from s. We can find a fractional flow of this value as follows: saturate all the outgoing edges from s. From each node i there are $|G_i|$ edges to nodes in W. Send $\gamma_i / \left\lfloor p\frac{|D_i|}{\beta} \right\rfloor$ units of flow along each of the $|G_i|$ outgoing edges from i. It is easy to see that $\gamma_i / \left\lfloor p\frac{|D_i|}{\beta} \right\rfloor \leq \frac{\beta}{p} - 1$, and therefore we do not violate the capacity constraints on edges from U to W. Observe that the total incoming flow to a vertex in W is at most $\beta - p$ since there are at most p incoming edges, each carrying at most $\frac{\beta}{p} - 1$ units of flow. An integral max flow in this network will correspond to γ_i units of flow going from s to i, and from i to all vertices in G_i before reaching t. If f units of flow fare sent from node $i \in U$ to node $j \in W$ means that disk j will send item i to f disks in $D_i \setminus G_i$.

Construct a transfer graph, similar to the method stated in Lemma 5.2.3, to satisfy all disks in $D_i \setminus G_i$. As in Lemma 5.2.4, the in-degree of this transfer graph is at most β. For each disk which belongs to some G_i, its out-degree is at most $\beta - p$. Among all disks in D_i, $\left\lfloor p\frac{|D_i|}{\beta} \right\rfloor$ disks are satisfied in Step 2 since they belong to G_i, and G_i can satisfy $\left\lfloor \left\lfloor p\frac{|D_i|}{\beta} \right\rfloor (\frac{\beta}{p} - 1) \right\rfloor$ disks in Step 3. The number of disks that still need item i are:

$$|D_i| - \left\lfloor p\frac{|D_i|}{\beta} \right\rfloor - \left\lfloor \left\lfloor p\frac{|D_i|}{\beta} \right\rfloor (\frac{\beta}{p} - 1) \right\rfloor = |D_i| - \left\lfloor \left\lfloor p\frac{|D_i|}{\beta} \right\rfloor \frac{\beta}{p} \right\rfloor \leq |D_i| - \left\lfloor |D_i| - \left\lceil \frac{\beta}{p} \right\rceil \right\rfloor = \left\lceil \frac{\beta}{p} \right\rceil .$$

Source s_i is responsible for all these disks. Therefore the out-degree of s_i is at most $\left\lceil \frac{\beta}{p} \right\rceil$, and the total out-degree of a node is at most $(\beta - p) + \left\lceil \frac{\beta}{p} \right\rceil$.

Similar to Lemma 5.2.4, each disk can be a source for up to $p + 1$ items, because it can be the original source of item i, and it also belongs to p different G_k $(k \neq i)$ sets. Thus there are upto $p + 1$ directed edges in each direction. $\square$

Theorem 5.3.4 *The total number of rounds is* $\max_i \log \left\lfloor p\frac{|D_i|}{\beta} \right\rfloor + 2\beta + \left\lceil \frac{\beta}{p} \right\rceil + 4p + 6$. *When* p *is*

$\Theta(\sqrt{\beta})$*, the total number of rounds is minimized, and is equal to* $\max_i \log |D_i| + 2\beta + O(\sqrt{\beta})$*.*

Proof The number of rounds taken in Step 3 is $2\beta + \left\lceil \frac{\beta}{p} \right\rceil + p + 2$ from Lemma 5.3.3 and Theorem 5.1.1. Combined with Lemma 5.3.2, the first result can be easily obtained. The second result is obtained by substituting p with $\Theta(\sqrt{\beta})$. $\square$

As $\max_i \log |D_i|$ and β are lower bounds of the problem, from Theorem 5.3.4 and as β goes to infinity, we have a polynomial-time $3 + o(1)$-approximation algorithm.

5.4 Allowing Bypass Disks

The main idea is that without bypass disks, only a small fraction of the N disks are included in G_i for some i, if one disk requests many items while, on average, each disk requests few items. If we allow bypass disks then we do not require that G_i is a subset of D_i. With bigger G_i sets, we can reduce the out-degree of the transfer graphs and thus reduces the total number of rounds.

Algorithm Multi-Source Multicast Allowing Bypass Disks

1. We define $\overline{\beta}$ as $\frac{1}{N} \sum_{i=1 \ldots N} |\{j | i \in D_j\}|$. In other words, $\overline{\beta}$ is the average number of items a disk requires, averaging over all disks. We arbitrarily choose a disjoint collection of subsets G_i, $i = 1 \ldots \Delta$ with a constraint that $|G_i| = \left\lfloor \frac{|D_i|}{\lceil \overline{\beta} \rceil} \right\rfloor$. By allowing bypass disks, G_i is not necessarily a subset of D_i.

2. This is the same as Step 2 in the Multi-Source Multicast Algorithm, except that the source for item i (namely disk i) may belong to G_j for some j.

3. This step is similar to Step 3 in the Multi-Source Multicast Algorithm. We add $\lceil \overline{\beta} \rceil$ edges from each disk in G_i to satisfy $\lceil \overline{\beta} \rceil \cdot \left\lfloor \frac{|D_i|}{\lceil \overline{\beta} \rceil} \right\rfloor$ disks in D_i, and add at most another $\lceil \overline{\beta} \rceil - 1$ edges from disk i to satisfy the remaining disks in D_i.

4. This is the same as Step 4 of the Multi-Source Multicast Algorithm.

Theorem 5.4.1 *The total number of rounds required for the multi-source multicast algorithm, by allowing bypass disks, is* $\max_i \lceil \log |D_i| \rceil + \beta + \lceil 2\overline{\beta} \rceil + 6$.

Proof The analysis is very similar to the case without bypass disks and here we only highlight the differences. We now show that the total size of the sets G_i is at most N.

$$\sum_i |G_i| \leq \sum_i \frac{|D_i|}{\lceil \overline{\beta} \rceil} \leq \frac{1}{\overline{\beta}} \sum_i |D_i|.$$

Note that $\sum_i |D_i|$ is $\overline{\beta}N$ by the definition of $\overline{\beta}$. This proves the upper bound of N on the total size of all the sets G_i. Step 2 takes $\max_i \lceil \log |D_i| \rceil + 2$ rounds. Note that this is 1 round larger than the bound in Lemma 5.2.2 as $\lceil \overline{\beta} \rceil$ can be 1. The in-degree of any disk in the transfer graph is still at most β, while the out-degree of any disk in the transfer graph is at most $\lceil \overline{\beta} \rceil + (\lceil \overline{\beta} \rceil - 1)$. The multiplicity of the graph is still at most 4. Thus, the total number of rounds is $(\max_i \lceil \log |D_i| \rceil + 2) + \beta + \lceil \overline{\beta} \rceil + (\lceil \overline{\beta} \rceil - 1) + 4 \leq \max_i \lceil \log |D_i| \rceil + \beta + \lceil 2\overline{\beta} \rceil + 6$. □

We now argue that $\lceil 2\overline{\beta} \rceil$ is a lower bound on the optimal number of rounds. Intuitively, on average, every disk has to spend $\overline{\beta}$ rounds to send data, and another $\overline{\beta}$ rounds to receive data. As a result, the total number of rounds cannot be smaller than $\lceil 2\overline{\beta} \rceil$. This can be seen by simply computing the total number of required transfers, and dividing by the number of transfers that can take place in each round. Allowing bypass disks does not change the fact that $\max(\max_i \lceil \log |D_i| \rceil, \beta)$ is the other lower bound. Therefore, we have a 3-approximation algorithm.

5.5 Bounded-Size Matching Model

So far, we assume communications can be performed on any matchings of the disks in each round. This model assumes unbounded bandwidth in the network. In this section we consider *bounded-size matching model*, where only a limited number of transfers are allowed in each round. We have a method to convert any constant factor approximation algorithm for the full matching model to a constant factor approximation algorithm for the bounded-size matching model. The only constraint is that no bypass disks are allowed.

Suppose that at most B transfers are allowed in each round, the method is as follows. Let E_i be the transfers in i-th round in the algorithm for the full matching model. Then we split each E_i into $\lceil |E_i|/B \rceil$ sets of size at most B and perform each set in a round. Let us denote the number of rounds required in an optimal solution for the full matching model and bounded-size matching model as OPT and OPT', respectively.

Theorem 5.5.1 *Given a ρ-approximation algorithm for the full matching model, we have $1 + \rho(1-1/B)$-approximation algorithm for the bounded-size matching model, where B is the maximum number of transfers allowed in a round.*

Proof Denote the number of rounds in our algorithm as t and t'. Note that since we move data only to disks that need the data, the total number of data transfers performed by the algorithm is the minimum possible. Thus $OPT' \geq \sum_i |E_i|/B$. Since $t \leq \rho OPT$ and $OPT \leq OPT'$, we have $t \leq \rho OPT'$.

Therefore,

$$\begin{aligned} t' &= \sum_{i=1}^{t} \left\lceil \frac{|E_i|}{B} \right\rceil \\ &\leq \sum_{i=1}^{t} \left(\frac{|E_i| - 1}{B} + 1 \right) \\ &= \frac{1}{B} \sum_{i=1}^{t} |E_i| + t \left(1 - \frac{1}{B} \right) \\ &\leq OPT' + \rho OPT' \left(1 - \frac{1}{B} \right) \\ &= \left(1 + \rho \left(1 - \frac{1}{B} \right) \right) OPT' \end{aligned}$$

□

Similarly, we can obtain the following theorem.

Theorem 5.5.2 *Given an approximation algorithm that takes at most $OPT + k$ rounds for the full matching model, we have an approximation algorithm that takes at most $(2 - \frac{1}{B})OPT' + k(1 - \frac{1}{B})$ rounds for the bounded-size matching model, where B is the maximum number of transfers allowed in a round.*

When combined with the results in previous chapters, we have constant factor approximation algorithms for the single-source multicast, multi-source broadcast, and multi-source multicast problems.

Corollary 5.5.3 *We have an approximation algorithm that takes at most* $(2-1/B)OPT+\Delta(1-1/B)$ *rounds for the single-source multicast problem using the bounded-size matching model.*

Corollary 5.5.4 *We have an approximation algorithm that takes at most* $(2-1/B)OPT+3(1-1/B)$ *rounds for the multi-source broadcast problem using the bounded-size matching model.*

Corollary 5.5.5 *We have a* $(1+(3+o(1))(1-1/B))$*-approximation algorithm for the multi-source multicast problem using the bounded-size matching model.*

5.6 NP-hardness Result

We will prove the multi-source multicasting problem to be NP-hard by showing a reduction from a restricted version of 3SAT. Papadimitriou [79] showed that 3SAT remains NP-complete even for expressions in which each variable is restricted to appear at most three times, and each literal at most twice. We denote this problem as 3SAT(3).

We assume that each literal appears at least once in the given instance. If not, we can always simplify the instance so that each literal appears at least once.

Given a 3SAT(3) instance, we create a multi-source multicast instance such that the 3SAT(3) instance is satisfied if and only if the corresponding multi-source multicast instance can transfer all items in 3 rounds.

Part I. For each variable x_i, we create (i) a source disk having item x_i, (ii) a set of destination disks X_i of size 3 which need item x_i, (iii) a source disk having item $\overline{x}_i$, (iv) a set of destination disks $\overline{X}_i$ of size 3 which need item $\overline{x}_i$, (v) a source disk having item s_i, (vi) a disk w_i (we call it a switch disk) which wants to receive items x_i, $\overline{x}_i$ and s_i, and (vii) 6 disks which need item s_i.

Part II. For each clause j, we create (i) a source disk having item c_j, and (ii) a set of destination disks C_j of size 2 (the size should be 4 instead, if there are only two literals in clause j) that need item c_j. Moreover, for each literal in clause j, arbitrarily pick one disk in the set of destination disks corresponding to the literal, and that disk, which originally only needs the item corresponding to the literal, will also need item c_j. For example, if clause j is $x_p \vee \overline{x}_q \vee x_r$, then one disk d in X_p, one disk in $\overline{X}_q$ and one disk in X_r, need item c_j. If there is another clause j' contains literal x_p, we pick one disk in $X_p \setminus \{d\}$ and that disk now needs item j'.

Lemma 5.6.1 *If the 3SAT(3) instance is satisfiable, there exists a valid schedule to finish all data transfers in* 3 *rounds.*

Proof It is easy to see that all seven disks demanding item s_i can be scheduled in three rounds. In particular, we schedule switch disk w_i to receive s_i in round 3 for all i. If variable x_i is **true**, we schedule switch disk w_i to receive $\overline{x}_i$ and x_i in round 1 and 2 respectively. x_i can be sent to a disk in X_i in round 1, making X_i receive items faster than $\overline{X}_i$. After round 2, two disks in X_i received item x_i, while only 1 disk in $\overline{X}_i$ received item $\overline{x}_i$. In round 3, the source disk of x_i can satisfy the last disk in X_i which has not received x_i. Note that the remaining two disks in X_i are idle and they can receive item c_j from other disks. Furthermore, since a disk in X_i gets item x_i in round 1 (but not in round 2), it is not difficult to see that the two disks in X_i can receive item c_j from other disks in either round 2 or round 3, and still all requests in X_i can be satisfied. On the other hand, the remaining two disks in $\overline{X}_i$ can be satisfied in round 3 by the source and one disk in $\overline{X}_i$. Note that all disks in $\overline{X}_i$ and the source of $\overline{x}_i$ are busy in this round. Thus, all requested items appeared in **Part I** are satisfied. If the variable is **false**, we schedule the switch disk to receive x_i in round 1, then $\overline{x}_i$ in round 2. As a result, two disks in $\overline{X}_i$ are idle in round 3, while all disks in X_i are busy in round 3.

We claim that both disks in C_j, for all j, can be satisfied as well. For example, if clause j is $x_p \vee \overline{x}_q \vee x_r$, and suppose x_p is **true** in a satisfying assignment. From the argument above, there exists a schedule such that the disk in X_p, which needs x_p and c_j, is idle in round 3. However, if $\overline{x}_q$ and x_r are **false**, the disk in $\overline{X}_q$, which needs $\overline{x}_q$ and c_j, and the disk in X_r, which needs x_r and c_j, are busy getting an item $\overline{x}_q$ and item x_r, respectively, in round 3. Even when $\overline{x}_q$ or x_r is **true**, we can schedule the transfers of item $\overline{x}_q$ and x_r such that the disk in $\overline{X}_q$, which needs $\overline{x}_q$ and c_j, or the disk in X_r, which needs x_r and c_j, is busy getting an item in round 3. We can do this because if variable x_i is **true**, two disks in X_i can receive item c_j at either round 2 or round 3. A valid schedule can send item c_j from the source to one disk in C_j in round 1. In round 2, we now have two copies of c_j to satisfy disks in $\overline{X}_q$ and X_r. In round 3, without the help of disks in $\overline{X}_q$ and X_r, we can satisfy 2 more disks, namely the second disk in C_j and the disk in X_p. If there are only two literals in clause j, the argument is similar. We need to satisfy two more disks in C_j, but, in round 2, we need to satisfy only one disk, instead of two disks, which cannot contribute item c_j in round 3. Thus, all requested items that appeared in **Part II** are satisfied as well. $\square$

Lemma 5.6.2 *If there is a valid schedule to finish all data transfers in 3 rounds, then the 3SAT(3) instance is satisfiable.*

Proof Since there are 7 disks that need item s_i, if we have to finish all transfers in 3 rounds, once a disk receives s_i, it will be busy until round 3. Note that all switch disks have to receive s_i, x_i and $\overline{x}_i$. Therefore, all switch disks have to receive item x_i and $\overline{x}_i$ in the first two rounds, and s_i in round 3. If switch disk i receives item x_i in round 1, we set literal $\overline{x}_i$ to be **true**. Otherwise, we set literal x_i to be **true**. Consider the former case: disks in X_i receive item x_i starting at round 2, meaning that all disks in X_i should be busy in round 3 to send or receive x_i. Suppose literal x_i appears in clauses j and k. Two disks in X_i may have to receive item c_j and c_k in the first 2 rounds. Thus, our construction restricts that if a literal x_i is set to **false**, disks in X_i cannot receive item c_j in round 3.

Consider a clause j, for instance, $x_p \vee \overline{x}_q \vee x_r$, a disk in X_p, a disk in $\overline{X}_q$, a disk in X_r, and both disks in C_j need item c_j. If all three literals are **false**, it is possible to satisfy the first three disks in the first 2 rounds. However, since all these three disks are busy in round 3, the source of c_j cannot satisfy both disks in C_j, which is a contradiction. Therefore, clause j has at least one true literal. If there are only two literals in clause j, the argument is similar. Because C_j is larger, all requests of item c_j cannot be satisfied when both literals are **false**. $\square$

Theorem 5.6.3 *The multi-source multicasting problem is NP-hard*

Proof It is easy to see that the reduction is polynomial, and together with Lemma 5.6.1 and Lemma 5.6.2, we conclude that the problem is NP-hard. $\square$

Chapter 6

Broadcasting in Two-tier Communication Networks

In this chapter[1], we study problems of broadcasting and multicasting in two-tier communication networks, which arises in Networks of Workstations [80, 4], grid computing [33], and clustered wide-area network systems [64, 63, 15]. We first give an approximation algorithm for this problem. Using this algorithm as a building box, we give an approximation algorithm for the multicast problem. We also give algorithms for the broadcast and multicast problems where the first-tier network does not have unlimited bandwidth. We then consider the *postal model* [9] version of the problems and give approximation algorithms. Lastly, we present an experimental study of the effect of having inaccurate information regarding the sizes of the clusters.

6.1 Problem Specification

We assume we have k clusters of processors. Cluster K_i has size $n_i, i = 0 \ldots (k-1)$, the number of processors in the i-th cluster. The total number of processors, denoted by N, is $\sum_{i=0}^{k-1} n_i$. We will assume that the broadcast/multicast originates at a processor in K_0. We order the *remaining* clusters in non-increasing size order. Hence $n_1 \geq n_2 \geq \ldots \geq n_{k-1}$. Clearly, n_0 could be smaller or larger than n_1, since it is simply the cluster that originates the broadcast/multicast.

In the broadcast problem, the goal is to minimize the number of rounds needed to convey the message from the source processor in K_0 to all other processors. In the multicast problem, the goal is to minimize the number of rounds needed to convey the message from the source processor in K_0 to a specified subset of processors, while the remaining processors may help to forward the message for others.

A message may be sent from a processor, once it has received the message. If the message is sent to a processor in its cluster, the message arrives one time unit later. If the message is sent

[1]This is joint work with S. Khuller and Y. Kim [60].

to a processor in a different cluster then the message arrives C time units later. Both the sending and receiving processors are busy during those C time units. In addition, in our algorithms we assume that each cluster advertises a single address to which messages are sent. Each cluster thus receives a message only once at this processor and then the message is propagated to different processors in the cluster. Thus new processors may be added or dropped without having to inform other clusters of the exact set of new addresses (we only need to keep track of the sizes of the clusters). Another advantage of this approach is that the total number of global communications, which are often costly, is minimized. In some cases, the broadcast time can be reduced by having many messages arrive at the same cluster. However, when we compare to the optimal solution we do not make any assumptions about the communication structure of the optimal solution.

In Section 6.5 we consider a slightly different model, the postal model, where a processor is busy for only one time unit when it sends a message. The time a message arrives at a receiver depends on whether the sender and receiver are in the same cluster or not— it takes one time unit if it is a local transfer, and C time units otherwise.

6.2 Broadcasting

The high level description of the algorithm is as follows. The source node first performs a local broadcast within its cluster. This takes $\lceil \log n_0 \rceil$ rounds. After all the nodes of K_0 have the message, we broadcast the message to the first n_0 clusters. Each node in K_0 sends a message to a distinct cluster. This takes exactly C rounds. Each cluster that receives a message, does a local broadcast within its cluster. All nodes that have received the message then send the message to distinct clusters. Again this takes C more rounds. While doing this, every node in K_0 keeps sending a message to a cluster that has not received a message as yet. Repeat this until all the processors receive the message. We call this algorithm *Largest Cluster First (LCF)* as we always choose the largest cluster as a receiver among clusters that have not received the message.

Algorithm *LCF*

1. Broadcast locally in K_0 (this takes $\lceil \log n_0 \rceil$ rounds).

2. Each cluster performs the following until all processors get informed.

   (a) cluster K_i in which all processors have messages, picks the first n_i clusters that have not received a message, and sends the message to them at every C time unit. Repeat until all clusters have at least one message.

   (b) Each cluster which received a message does local broadcasting until all processors in the cluster have messages[2].

Note that in our algorithm each cluster receives only one message from other clusters. That is, the total number of global transfers is minimized (we need $k-1$ global transfers). This property is important since we want to avoid consuming unnecessary wide-area bandwidth which is usually expensive.

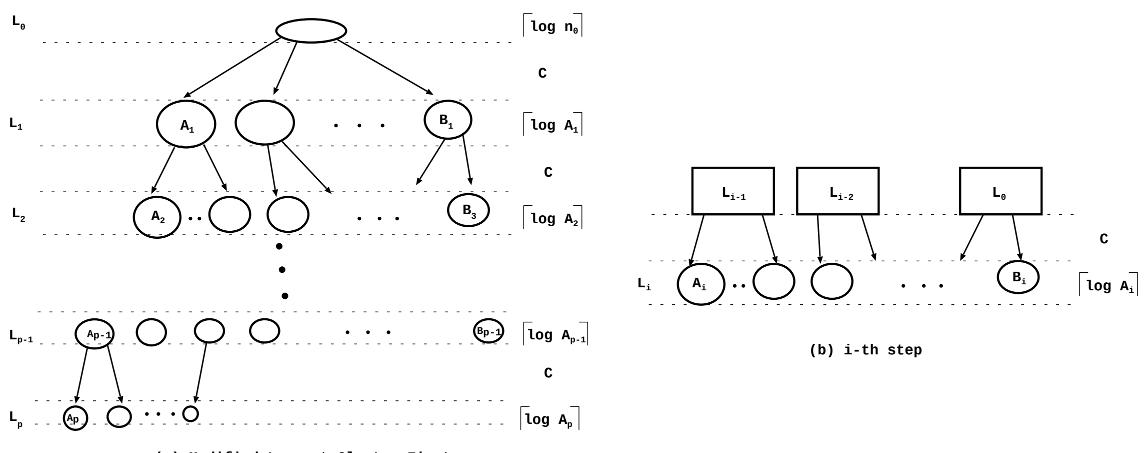
6.2.1 Analysis

In this subsection, we prove that *LCF* gives a 2-approximation. For the purpose of analysis, we modify *LCF* slightly. The makespan of the schedule by the *modified* algorithm may be worse than the original algorithm (but no better) and it is at most 2 times the optimal.

In *Modified LCF*, local and global phases take place in turn (see Figure 6.1). Let L_i be the set of clusters that receive the message at the i-th global step. For example, L_0 includes K_0 and L_1 includes all clusters that receive the message from K_0 at the end of the first global phase. Let N_i be the total number of processors in clusters belonging to L_i. That is, $N_i = \sum_{K \in L_i} |K|$. At the i-th step, all processors in clusters $K \in L_j$ ($j = 0, \ldots, i-1$) send messages to L_i and then clusters in L_i perform local broadcasting. Therefore, the i-th step takes $C + \lceil \log A_i \rceil$ rounds.

Algorithm *Modified LCF*

[2]We interrupt all local broadcasting and do one global transfer, if the number of processors having the message is at least the number of clusters that have not received a message.



(a) Modified Largest Cluster First

(b) i-th step

Figure 6.1: (a) Local and global phases in *Modified LCF* take place in turn (b) At the i-th step, all processors in clusters $K \in L_j$ $(j = 0, \dots, i-1)$ send messages to L_i and then clusters in L_i perform local broadcasting.

1. Broadcast locally in K_0. Then we have that $L_0 = K_0$ and $N_0 = n_0$.
2. At the i-th step (repeat until all processors get informed)
   (a) Global phase: Pick $\sum_{j=0\dots i-1} N_j$ largest clusters that are not informed as yet. Each processor in $\bigcup_{j=0\dots i-1} L_j$ sends one message to each of those clusters.
   (b) Local phase: Clusters in L_i do local broadcasting.

Let p be the number of global transfer steps that *Modified LCF* uses. Then we have the following theorem.

Theorem 6.2.1 *The broadcast time of our algorithm is at most* $2 \log N + pC + 3$.

Define A_i (B_i) to be the biggest (smallest) cluster in L_i. We need the following two lemmas to prove this theorem.

Lemma 6.2.2 *For* $i = 0 \dots p-1$, $n_0 \cdot |B_1| \cdots |B_i| \leq N_i$.

Proof We prove this by induction. For $i = 0$, it is true since $N_0 = n_0$. Suppose that for $i = l$ $(< p-1)$ we have $n_0 \cdot |B_1| \cdots |B_l| \leq N_l$. Since at the $(l+1)$-th global transfer step, every node in N_l will send the message to a cluster in L_{l+1}, $|L_{l+1}| \geq N_l$. Furthermore, the size of clusters in L_{l+1} is at least $|B_{l+1}|$ by definition. Therefore,

$$\begin{aligned} N_{l+1} &= \sum_{K \in L_{l+1}} |K| \\ &\geq \sum_{K \in L_{l+1}} |B_{l+1}| \\ &= |L_{l+1}| \cdot |B_{l+1}| \\ &\geq N_l \cdot |B_{l+1}| \\ &\geq n_0 \cdot |B_1| \cdots |B_l| \cdot |B_{l+1}|. \end{aligned}$$

□

Lemma 6.2.3 $\log |A_1| < \log N - (p-2)$.

Proof After we have all processors in A_1 receive the message we need $p-1$ more global transfer steps. With $|A_1|$ copies, we can make $|A_1| \cdot 2^i$ processors receive the message after i global transfer steps by doubling the number of copies in each global step. Therefore $|A_1| \cdot 2^{p-2} < N$ (otherwise, we do not need the p-th global broadcasting step). □

Proof of Theorem 6.2.1. The upper bound of the total broadcast time for local transfer phases is $\lceil \log n_0 \rceil + \lceil \log |A_1| \rceil + \ldots + \lceil \log |A_p| \rceil$. Since we have $|A_i| \leq |B_{i-1}|$ (for $2 \leq i \leq p$) in *LCF*, it is upper bounded by $\lceil \log n_0 \rceil + \lceil \log |A_1| \rceil + \lceil \log |B_1| \rceil + \ldots + \lceil \log |B_{p-1}| \rceil$. By Lemma 6.2.2 and Lemma 6.2.3, the total broadcast time only for local transfer steps is at most

$$\begin{aligned} & \lceil \log n_0 \rceil + \lceil \log |A_1| \rceil + \lceil \log |B_1| \rceil + \ldots + \lceil \log |B_{p-1}| \rceil \\ \leq \; & \log n_0 + \log |B_1| + \ldots + \log |B_{p-1}| + \log |A_1| + p + 1 \\ < \; & \log n_0 \cdot |B_1| \cdots |B_{p-1}| + \log N - (p-2) + p + 1 \\ \leq \; & \log N_{p-1} + \log N + 3 \\ \leq \; & 2 \log N + 3. \end{aligned}$$

Our schedule uses p global transfer steps, taking a total of pC additional rounds. □

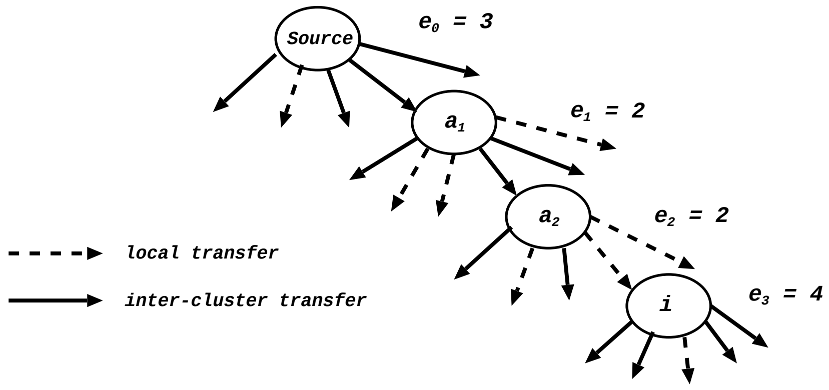


Figure 6.2: An example to show the inter-cluster transfers processor i experiences.

In the next lemma, we prove that pC is a lower bound on the optimal broadcast time. To prove this we count the number of inter-cluster transfers a processor *experiences* as follows. Given a broadcast schedule, let the path from the source to the processor i, be $a_0, a_1, \ldots, a_l = i$. That is, the source a_0 sends the message to a_1 and a_1 sends to a_2 and so on. Finally a_{l-1} sends the message to processor i $(= a_l)$. Let e_j $(j = 0 \ldots l-1)$ represent the number of processors that receive the message from a_j via inter-cluster transfers until a_{j+1} receives the message (including the transfer to a_{j+1} if they are in different clusters). In addition, let e_l be the number of processors that receive the message from processor i via inter-cluster transfers. That is, i sends the message to e_l processors in other clusters. Then we say processor i experiences e inter-cluster transfers where $e = \sum_{j=0}^{l} e_j$. Figure 6.2 shows an example of how to count the number of inter-cluster transfers that a processor experiences. In the example, processor i experiences $3 + 2 + 2 + 4 = 11$ inter-cluster transfers. If there is any processor that experiences p inter-cluster transfers, then pC is a lower bound on the optimal solution.

Lemma 6.2.4 *At least one node in the optimal solution experiences p inter-cluster transfers.*

Proof Imagine a (more powerful) model in which once a node in a cluster receives the message, all nodes in the cluster receives the message instantly (That is, local transfers take zero unit of time). In this model the broadcast time is given by the maximum number of inter-cluster transfers that any processor experiences. We will prove that *LCF* gives an optimal solution for this model. Since *LCF* uses p global transfer steps, the optimal broadcast time is pC in this model. Since this lower bound is for a stronger model, it also works in our model.

Suppose that there is a pair of clusters K_i and K_j $(0 < i < j < k)$ such that K_j receives the message earlier than K_i in the optimal solution. Let K_i receive the message at time t_i and K_j receive at time t_j $(t_i > t_j)$. Modify the solution as follows. At time $t_j - C$ we send the message to K_i instead of K_j and K_i performs broadcasting as K_j does until t_i (This can be done since the size of K_i is at least as big as K_j.) At time t_i, K_j receives the message and after that the same schedule can be done. This exchange does not increase the broadcast time and therefore, *LCF* gives an optimal solution for the model with zero local transfer costs.

We now prove the lemma by contradiction. Suppose that there is an optimal solution (for the original model) in which all processors experience at most $p-1$ inter-cluster transfers. Then in the model with zero local transfer costs we should be able to find a solution with broadcasting time $(p-1)C$ by ignoring local transfers, which is a contradiction. $\square$

Using the above lemma, we know that pC is a lower bound on the optimal broadcast time. Since $\log N$ is also a lower bound on the optimal solution, this gives us 3-approximation (and an additive term of 3). However, the lower bound pC considers only the global communications of the optimal solution. On the other hand, the lower bound $\log N$ only counts the local transfers. To get a better bound, we prove the following theorem that combines the lower bounds developed above.

Theorem 6.2.5 *The optimal solution has to take at least* $(p-1)(C-1) + \lceil \log \frac{N}{2} \rceil$ *rounds.*

Proof Consider an optimal schedule, in the end all N nodes receive the message. We partition all nodes into two sets, S_l and S_s, where S_l contains all nodes which experienced at least $p-1$ inter-cluster transfers, and S_s contains all nodes which experienced at most $p-2$ inter-cluster transfers. We now show that $|S_s| < \frac{N}{2}$. Suppose this is not the case, it means that the optimal solution can satisfy at least $\frac{N}{2}$ nodes using at most $p-2$ inter-cluster transfers. Using one more round of transfers, we can double the number of nodes having the message and satisfy all N nodes. This is a contradiction, since we use less than p inter-cluster transfers. Therefore we have $|S_l| \geq \frac{N}{2}$. Since originally we have one copy of the message, satisfying nodes in S_l takes at least $\lceil \log \frac{N}{2} \rceil$ transfers. So at least one node in S_l experienced $\lceil \log \frac{N}{2} \rceil$ transfers (either inter-cluster or local transfers). We know that all nodes in S_l experienced at least $p-1$ inter-cluster transfers. The node needs at least $(p-1)C + (\lceil \log \frac{N}{2} \rceil - (p-1))$ rounds to finish. □

We now prove a central result about Algorithm *LCF* which will be used later.

Theorem 6.2.6 *Our algorithm takes at most $2OPT+7$ rounds. Moreover, it takes at most $2OPT$ rounds when both p and C are not very small (i.e., when $(p-2)(C-2) \geq 7$).*

Proof If C is less than 2, we can treat the nodes as one large cluster and do broadcasting. This takes at most $C\lceil \log N \rceil$ and is a 2-approximation algorithm. The problem is also trivial if p is 1, because in this case $n_0 \geq k-1$. Therefore we consider the case where both values are at least 2. Here we make use of Theorems 6.2.1 and 6.2.5.

$$\begin{aligned}
& (2OPT+7) - (2\log N + pC + 3) \\
\geq \ & \left(2\left((p-1)(C-1) + \left\lceil \log \frac{N}{2} \right\rceil\right) + 7\right) - (2\log N + pC + 3) \\
\geq \ & (p-2)(C-2) \\
\geq \ & 0 \qquad (\text{when } p \geq 2 \text{ and } C \geq 2).
\end{aligned}$$

□

Remark: Our algorithm takes at most $2OPT$ rounds when both p and C are not very small (i.e., when $(p-2)(C-2) \geq 7$).

Corollary 6.2.7 *We have a polynomial-time 2-approximation algorithm for the broadcasting problem.*

6.2.2 Bad Example

There are instances for which the broadcast time of *LCF* is almost 2 times the optimal. Suppose that we have 2 clusters, K_0 and K_1, each of size n_0 and n_1 ($n_0 \leq n_1$), respectively. In addition, there are $n_0 - 1$ more clusters, each of size 1. A node in K_0 has a message to broadcast. It is easy to see that the makespan of our algorithm is $\lceil \log n_0 \rceil + C + \lceil \log n_1 \rceil$. However, the broadcasting can be made faster by sending a message to K_1 before local broadcast in K_0 is finished. A possible schedule is (i) make one local copy in K_0 (ii) one processor in K_0 send a message to K_1 and another processor does local broadcast in K_0 (iii) after finishing local broadcast, processors in K_0 send messages to the remaining $n_0 - 1$ clusters (iv) clusters other than K_0 do local broadcasting as soon as they receive a message. The makespan of this solution is $1 + \max\{C + \lceil \log n_1 \rceil, \lceil \log(n_0 - 1) \rceil + C\}$. In the case where $n_0 \approx n_1$ and $\log n_0 \gg C$, the makespan of *LCF* is almost 2 times the optimal.

6.3 Multicasting

For multicasting, we need to have only a subset of processors receive the message. We may reduce the multicast time significantly by making use of large clusters that may not belong to the multicast group. Let n'_i denote the number of processors in K_i that belong to the multicast group. Let M denote the set of clusters (except K_0) in which some processor wants to receive the message and k' denote the size of set M. Formally, $M = \{K_i | n'_i > 0 \text{ and } i > 0\}$ and $k' = |M|$.

Let $LCF(m)$ be algorithm *LCF* to make m copies. That is, $LCF(m)$ runs in the same way as *LCF* but stops as soon as the total number of processors that received the message is at least m (we may generate up to $2(m-1)$ copies). For example, *LCF* for broadcasting is $LCF(N)$. Here is the algorithm.

```
Algorithm LCF Multicast
Phase 1: Run $LCF(k')$ by using any processor whether it belongs to the multicast group or not.
Phase 2: Send one copy to each cluster in $M$ if it has not received any message yet.
Phase 3: Do local broadcast in clusters of $M$.
```

6.3.1 Analysis

Let p' be the number of global transfer steps $LCF(k')$ uses. Suppose D be the number of rounds taken in the last local broadcast step in $LCF(k')$ (after the p'-th global transfer steps). Note that some nodes in clusters performing the last local broadcast may not receive the message, since we stop as soon as the total number of nodes having the message is at least k', and hence $D \leq \lceil \log |A_{p'}| \rceil$. Nevertheless, D may be greater than $\lceil \log |B_{p'}| \rceil$ and thus some clusters may stop local broadcast before the D-th round. Let $A_{p'+1}$ be the biggest cluster among clusters which have not received a copy after $LCF(k')$ has finished.

To get a 2-approximation algorithm, we need the following lemma, which bounds the sum of the number of rounds taken in the local phases in $LCF(k')$ and in the last local broadcast in clusters of M. The lemma still holds even when a cluster performing local broadcast in Phase 3 needs to broadcast to the whole cluster (i.e., $n'_i = n_i$).

Lemma 6.3.1 $(\log n_0 \cdot |B_1| \cdots |B_{p'-2}| + D) + \max(\lceil \log |A_{p'}| \rceil - D, \lceil \log |A_{p'+1}| \rceil) \leq \log k' + 2$

Proof Case I: $\lceil \log |A_{p'}| \rceil - D > \lceil \log |A_{p'+1}| \rceil$. It is easy to see that $(\log n_0 \cdot |B_1| \cdots |B_{p'-2}| + D) + (\lceil \log |A_{p'}| \rceil - D) \leq \log n_0 \cdot |B_1| \cdots |B_{p'-1}| + 1 \leq \log k' + 1$, and the lemma follows.

Case IIa: $\lceil \log |A_{p'}| \rceil - D \leq \lceil \log |A_{p'+1}| \rceil$ and $D > \lceil \log |B_{p'}| \rceil$. Note that $2^D \leq 2|A_{p'}| \leq 2|B_{p'-1}|$ and $|A_{p'+1}| \leq |B_{p'}|$; we have $(\log n_0 \cdot |B_1| \cdots |B_{p'-2}| + D) + (\lceil \log |A_{p'+1}| \rceil) < \log n_0 \cdot |B_1| \cdots |B_{p'}| + 2 \leq \log N_{p'-1} \cdot |B_{p'}| + 2$. After the p'-th global transfer step, one node in each of $N_{p'-1}$ clusters has just received the message. Each of these clusters will generate at least $|B_{p'}|$ copies (since $D > \lceil \log |B_{p'}| \rceil$), so $N_{p'-1} \cdot |B_{p'}| < k'$, and the lemma follows.

Case IIb: $\lceil \log |A_{p'}| \rceil - D \leq \lceil \log |A_{p'+1}| \rceil$ and $D \leq \lceil \log |B_{p'}| \rceil$. Note that $|A_{p'+1}| \leq |B_{p'-1}|$; we have $(\log n_0 \cdot |B_1| \cdots |B_{p'-2}| + D) + (\lceil \log |A_{p'+1}| \rceil) \leq \log n_0 \cdot |B_1| \cdots |B_{p'-1}| \cdot 2^D + 1 \leq \log N_{p'-1} \cdot 2^D + 1$. After the p'-th global transfer step, each cluster which has just received the message will generate at least 2^D copies, so $\log N_{p'-1} \cdot 2^{D-1} < k'$, and the lemma follows. □

Theorem 6.3.2 *Our multicast algorithm takes at most* $2\log k' + p'C + C + 4$ *rounds.*

Proof In a manner similar to the proof of Theorem 6.2.1, the broadcast time spent only in local transfer steps in $LCF(k')$ is at most

$$\begin{aligned} & \lceil \log n_0 \rceil + \lceil \log |A_1| \rceil + \ldots + \lceil \log |A_{p'-1}| \rceil + D \\ \leq \quad & \log n_0 + \log |B_1| + \ldots + \log |B_{p'-2}| + D + \log |A_1| + p' \\ < \quad & \log n_0 \cdot |B_1| \cdots |B_{p'-2}| + D + \log k' + 2. \end{aligned}$$

The second inequality holds because $\log |A_1| < \log k' - (p' - 2)$ by Lemma 6.2.3. Moreover, the global transfer steps in $LCF(k')$ and the second phase take $p'C$ and C rounds, respectively. Note that $n'_i \leq n_i$. In the third phase, all clusters which receive a message during the first phase need at most $\lceil \log |A_{p'}| \rceil - D$ rounds to do local broadcast. The remaining clusters, which receive a message during the second phase, are of size at most $|A_{p'+1}|$. Therefore local broadcasting takes at most $\lceil \log |A_{p'+1}| \rceil$ rounds. Using Lemma 6.3.1, we have the theorem. □

Lemma 6.3.3 *At least one node in the optimal solution experiences* p' *inter-cluster transfers.*

Proof The basic argument is the same as the one in Lemma 6.2.4. Note that $LCF(k')$ uses any processor whether it belongs to the multicast group or not. If the optimal solution does not use any processor that $LCF(k')$ uses, it cannot create new copies of the message faster than $LCF(k')$. □

Theorem 6.3.4 *The optimal solution takes at least* $(p' - 1)(C - 1) + \left\lceil \log \frac{k'}{2} \right\rceil$ *rounds.*

Proof The proof is very similar to the proof of Theorem 6.2.5. We partition all nodes into two sets, S_l and S_s. We now show that there are less than $\frac{k'}{2}$ distinct multicast clusters in S_s. Suppose this is not the case, it means that OPT can satisfy at least $\frac{k'}{2}$ distinct multicast clusters using at most $p' - 2$ inter-cluster transfers. Using one more round of transfers, all k' multicast clusters can receive the message, which is a contradiction. Therefore we have at least $\frac{k'}{2}$ distinct multicast clusters in S_l and $|S_l| \geq \frac{k'}{2}$. □

Theorem 6.3.5 *Our algorithm takes at most $2OPT + 10$ rounds. Moreover, it takes at most $2OPT$ rounds when both p' and C are not very small (i.e., when $(p' - 3)(C - 2) \geq 10$).*

Proof By making use of Theorems 6.3.2 and 6.3.4, and an analysis similar to that in Theorem 6.2.6, we can show that $(2OPT+10) - (2\log k' + p'C + C + 4) \geq (p' - 3)(C - 2)$. The problem is trivial when C is less than 2 or $p = 1$. When $p' = 2$, we can do an exhaustive search on the number of clusters in M which receives the message in the first global transfer step in $LCF(k')$. It is not difficult to prove that this also takes at most $2OPT + 10$ rounds. □

Remark: Our algorithm takes at most $2OPT$ rounds when both p' and C are not very small (i.e., when $(p' - 3)(C - 2) \geq 10$).

Corollary 6.3.6 *We have a polynomial-time 2-approximation algorithm for the multicast problem.*

6.4 Bounding Global Transfers

In the model we considered, we assume any node may communicate with any other node in other clusters, and the underlying network connecting clusters has unlimited capacity. A more practical model is to restrict the number of pairs of inter-cluster transfers that can happen simultaneously. In this section we present two models to restrict the network capacity. The *bounded degree model* restricts the number of inter-cluster transfers associated with a particular cluster. This model reflects the characteristics of a global network like the Internet, where the bottlenecks tend to occur at the edge of the global network. On the other hand, the *bounded-size matching model* restricts the total number of inter-cluster transfers at any given time. The bottleneck occurs at the core of the inter-cluster network, which is an appropriate model when homogeneous networks are arranged hierarchically.

6.4.1 Bounded Degree Model

Associate an additional parameter d_i with each cluster i, that limits the number of inter-cluster transfers from or to nodes in cluster i in a time unit. We call this limitation a degree constraint.

We denote an instance of this model be $I(n_i, d_i)$, meaning that there are n_i nodes in cluster K_i, and at most d_i of those may participate in inter-cluster transfers at any given time.

Algorithm *Bounded Degree Broadcast*

Given Instance $I(n_i, d_i)$, arbitrarily select a subset K_i' of d_i nodes in each cluster, and consider only the K_i'. We have a new instance $I(d_i, d_i)$. Note that $I(d_i, d_i)$ can be viewed as an instance of the general broadcast problem on the unrestricted model. In phase 1, run Algorithm *LCF* in Section 6.2 on $I(d_i, d_i)$. In phase 2, since there are d_i informed nodes in each cluster, we do local broadcasting to send the message to the remaining $n_i - d_i$ nodes.

An important observation is that since there is only a unique message, it does not matter which subset of nodes in a cluster perform inter-cluster transfers. What matters is the number of informed nodes in the clusters at any given time. The following lemma compares the optimal number of rounds taken by instances using the two different models.

Lemma 6.4.1 *The optimal schedule of Instance $I(d_i, d_i)$ takes no more rounds than the optimal schedule of the corresponding instance $I(n_i, d_i)$.*

Proof We argue that given an optimal schedule, which completes in OPT rounds, of Instance $I(n_i, d_i)$, we can create a schedule, which completes in at most OPT round, of the corresponding Instance $I(d_i, d_i)$.

Given an optimal schedule of Instance $I(n_i, d_i)$, let S_i be a set of the first d_i nodes in K_i that receive the message in the schedule. We can safely throw out all transfers (both inter-cluster and local transfers) in the schedule of which the receiving node is in $K_i \setminus S_i$, because we only need d_i nodes in $I(d_i, d_i)$. Let t_i be the time at which the last node in S_i receives the message. Consider any inter-cluster transfer which starts after t_i and is originated from a node in $K_i \setminus S_i$, we can modify the transfers so that it is originated from a node in S_i instead. It is not difficult to see that we can always find such a remapping, since there are at most d_i inter-cluster transfers at any given time, and nodes in S_i do not perform any local transfers after time t_i. Moreover, no node in $K_i \setminus S_i$ initiates a transfer on or before time t_i, because they have not received the message yet. Therefore we have removed all transfers involving nodes in $K_i \setminus S_i$. We can safely remove all the nodes in $K_i \setminus S_i$, effectively making the schedule applicable to Instance $I(d_i, d_i)$. Since we do not add any new transfers, the total number of rounds used cannot go up. Therefore the optimal number of rounds for $I(d_i, d_i)$ is at most that of $I(n_i, d_i)$. □

Theorem 6.4.2 *Our algorithm takes at most* $3OPT + 7$ *rounds.*

Proof Using Theorem 6.2.6 and Lemma 6.4.1, the first phase takes at most $2OPT + 7$ rounds. Moreover in phase 2, local broadcasting takes at most $\max_i \left\lceil \log \frac{n_i}{d_i} \right\rceil$ rounds, which is at most OPT. □

6.4.2 Bounded Degree Model: Multicasting

In this model only a subset M_i (possibly empty) of nodes in cluster K_i needs the message. Nodes in $K_i \setminus M_i$ may help passing the message around. Let n'_i be $|M_i|$. Observe that although we may make use of nodes in $K_i \setminus M_i$, we never need more than d_i nodes in each cluster, because of the degree constraint in the number of inter-cluster transfers. Similarly if $d_i \leq n'_i$, nodes in $K_i \setminus M_i$ are never needed. Therefore set n_i to be $\max(d_i, n'_i)$. Arbitrarily select d_i nodes for each cluster, with priority given to nodes in M_i. Run the LCF Multicast algorithm on the selected nodes. Now there are d_i nodes having the message in each cluster belongs to the multicast group, so we can do local broadcasting to satisfy the remaining nodes.

Theorem 6.4.3 *Our algorithm takes at most* $3OPT + 10$ *rounds.*

Proof We may run the LCF Multicast algorithm on the selected nodes, because there are at most d_i selected nodes from each cluster, all selected nodes can communicate each other freely as if there is no bounded degree constraint. Using the same idea as in Lemma 6.4.1, we can show that the optimal schedule of the instance given to the LCF Multicast algorithm takes no more rounds than the optimal schedule of the original bounded degree multicasting instance. We can apply the same idea because any schedule only need to use up to $n_i = \max(d_i, n'_i)$ nodes in each cluster for multicasting. By Theorem 6.3.5, the multicast steps takes $2OPT + 10$ rounds. Moreover, the local broadcasting phase only need to satisfy at most $n'_i - d_i$ nodes, which takes at most OPT rounds. $\square$

6.4.3 Bounded-Size Matching Model

In this model, we bound the number of inter-cluster transfers that can be performed simultaneously. Let us assume that we allow only B inter-cluster transfers at a time. Note that we can assume $B \leq \lfloor N/2 \rfloor$ since this is the maximum number of simultaneous transfers allowed by our matching-based communication model.

Algorithm *BoundedSizeBroadcast*

Phase 1: We run $LCF(B)$ to make B copies of the message.

Phase 2: Every C time units we make B more copies by inter-cluster transfers until all clusters have at least one copy of the message.

Phase 3: Do local broadcast to inform all the processors in each cluster.

Let p_B be the number of global transfer steps $LCF(B)$ uses, and p_L be the number of global transfer steps in the second stage of the algorithm.

Theorem 6.4.4 *We need* $2\log B + p_B C + p_L C + 4$ *rounds for broadcasting when we allow only* B *inter-cluster transfers at a time.*

Proof Note that the algorithm resembles the LCF Multicast algorithm. In phase 1 we use $LCF(B)$ instead of $LCF(k')$. In phase 2 we need p_L global transfer steps instead of 1 global transfer step to create at least one copy of the message in every cluster. In phase 3 the local broadcast always fills the entire cluster. (i.e., we can treat $n'_i = n_i$.) Despite the differences, we can use the same techniques to prove the stated bound. $\square$

Lemma 6.4.5 *In any schedule, there is a processor that experiences $p_B + p_L$ inter-cluster transfers.*

Proof The proof is similar to the proof of Lemma 6.2.4. In the model where local transfers take zero unit of time, the same exchange argument will show that the optimal broadcast time is $(p_B + p_L)C$.

If there is a schedule in which all processors experience less that $p_B + p_L$ inter-cluster transfers (in the original model), it is a contradiction to the assumption that the optimal broadcast time is $(p_B + p_L)C$ in the model with zero local transfer costs. □

Theorem 6.4.6 *The optimal solution takes at least $(p_B + p_L - 1)(C - 1) + \lceil \log B \rceil$ rounds.*

Proof The proof is very similar to the proof of Theorem 6.2.5. Note that in this case $|S_s| < N - B$, otherwise there is a way to satisfy all nodes using $p_B + p_L - 1$ rounds of inter-cluster transfers. Therefore $|S_l| \geq B$ and the theorem follows. □

Theorem 6.4.7 *Our algorithm takes at most $2OPT + 6$ rounds.*

Proof Using the proof technique in Theorem 6.2.6, the theorem follows from Theorem 6.4.4 and Theorem 6.4.6. □

Remark: Note that by setting $B = \lfloor N/2 \rfloor$, we can improve the makespan of the basic broadcasting (without any bound on the global transfers) by one round. This is because in this algorithm we stop performing local transfers when the number of copies is $\lfloor N/2 \rfloor$ (as more copies cannot contribute to global transfers) and start global transfers.

6.4.4 Bounded-Size Matching Model: Multicasting

In this model only a subset M_i of nodes in cluster K_i needs the message. Define $M = \{K_i | n'_i > 0 \text{ and } i > 0\}$ and $k' = |M|$. We assume $k' > B$ or otherwise we can use the *LCF Multicast* algorithm. We run $LCF(B)$ by using any processor available. Then every C time units we make B more copies by inter-cluster transfers until all clusters in M have at least one copy of the message. Lastly do local broadcast to inform all the processors in each cluster in M. Using the same techniques used to prove the bound on broadcasting under bounded-size matching model, we have the following theorem.

Theorem 6.4.8 *The algorithm takes at most* $2OPT + 10$ *rounds.*

6.5 Postal Model

In this section, we consider a slightly different model. This model is motivated by the interest in grid computing [33] and computing on clusters of nodes. In fact, the work on the MagPIe project [64, 63] specifically supports this communication model. In previous sections, we assumed that when processor p_i sends a message to processor p_j in another cluster, p_i is busy until p_j finishes receiving the message (this takes C time units). However, in some situations, it may not be realistic since p_i may become free after sending the message and does not have to wait until p_j receives the message.

In this section, we assume that a processor is busy for only one time unit when it sends a message. The time a message arrives at the receiver depends on whether the sender and receiver are in the same cluster or not—it takes one time unit if it is a local transfer (within the cluster), and C time units if it is an inter-cluster transfer.

We first show that *LCF* gives a 3-approximation in this model. In addition, we present another algorithm, which we call *Interleaved LCF*. Recall that we want to minimize the total number of global transfers as well as minimizing the makespan. Let OPT' denote the minimum makespan among all schedules that minimize the total number of global transfers. We can show that the makespan of the schedule generated by *Interleaved LCF* is at most 2 times OPT'.

6.5.1 Analysis of LCF

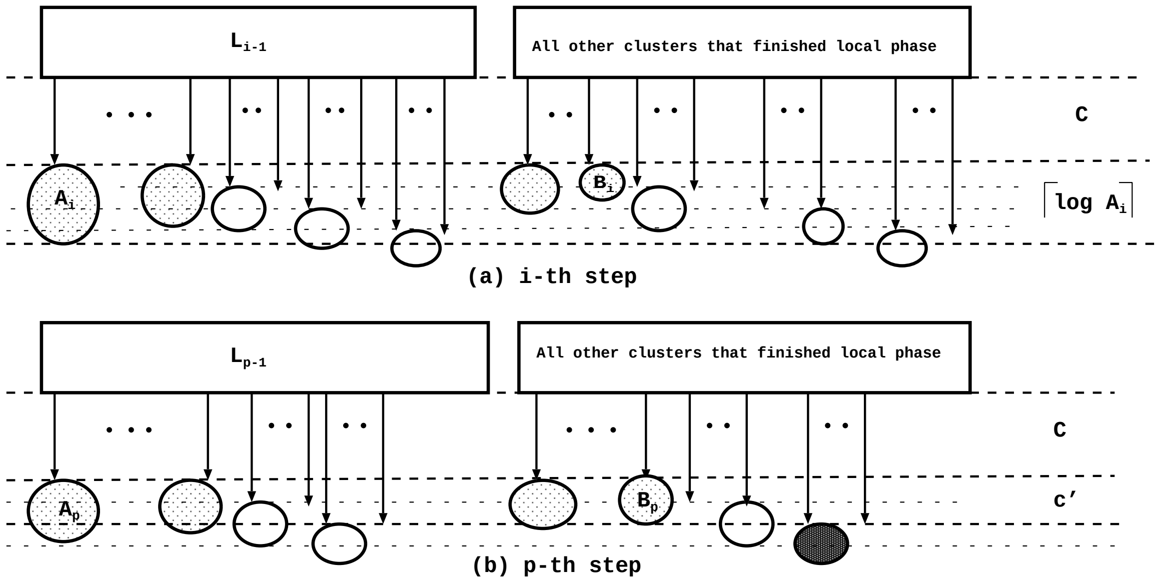
The analysis is similar to the one presented in Section 6.2.1. We modify the algorithm (for the analysis purpose) so that we have local and global phases in turn. However, in this model a processor can initiate more than one global transfer in a global phase since senders are busy for only one time unit per global transfer. We define A_i, B_i, L_i as follows. Let $A_0 = |K_0|$. After finishing local transfers in K_0, all processors in K_0 start global transfers. They can initiate a global transfer at every time unit. After C time units, n_0 clusters receive a message (denoted as L_1). Let $A_1(B_1)$ be the biggest(smallest) cluster among them. Now K_0 stops global transfers for $\lceil \log A_1 \rceil$ rounds (global transfers already initiated continue to be done). For those $\lceil \log A_1 \rceil$ rounds, every cluster that received the message performs (only) local broadcasting. For all the clusters that receive the message in this step, they have exactly $\lceil \log A_1 \rceil$ rounds of local broadcasting. So for example, if a cluster receives t time unit later than A_1 then it can only start its global transfers t time units later than A_1 (it will be idle even if it finishes local broadcasting earlier). Note that A_1 is the biggest cluster in L_1 and therefore, $\lceil \log A_1 \rceil$ is enough for local broadcasting of those clusters. After $\lceil \log |A_1| \rceil$ time units, we have all clusters that have finished local broadcasting phase perform global transfers every time unit *for C rounds.* Clusters that have not finished local phase keep performing local broadcasting (or wait) and then start global transfers. Repeat this until all processors get informed. In general, we define L_i as clusters that receive messages in the first global transfers of the i-th step (so they can participate in global phase of the $(i+1)$-th step from the beginning). Figure 6.3(a) shows the i-th step. In this example $C = 4$ so a processor can initiate (at most) 4 global transfers. Dotted clusters belong to L_i. A_i (B_i) is the biggest (smallest) cluster in L_i. Suppose that the schedule has p global phases. Then it is easy to see that Lemmas 6.2.2 and 6.2.3 hold. There is one subtle case where there are some clusters that receive the message later than A_p by the transfers initiated in the p-th global phases (see Figure 6.3(b)).

We first analyze the makespan of the simple case where A_p is one of the last clusters that receive the message.

Theorem 6.5.1 *The makespan of* Modified LCF *is at most* $2 \log N + pC + 3$ *when* A_p *is one of the last clusters that receive the message, and* $2 \log N + pC + c' + 3$ *when there are some clusters that receive the message* c' *time units later than* A_p*.*

Proof The total makespan taken for local transfers is

$$\lceil \log n_0 \rceil + \lceil \log |A_1| \rceil + \cdots + \log \lceil |A_p| \rceil$$



L_{i-1}
All other clusters that finished local phase
C
A_i
B_i
⌈log A_i⌉
(a) i-th step
L_{p-1}
All other clusters that finished local phase
C
C′
A_p
B_p
(b) p-th step

Figure 6.3: (a) The figure shows the i-th step of *LCF* under the postal model. Dotted clusters belong to L_i. (b) In the p-th step, there can be some processors that receive messages later than A_p. The dark circle is the last cluster that receives the message

$$
\begin{aligned}
&\leq \log n_0 + \log|A_1| + \cdots + \log|A_p| + (p+1) \\
&\leq \log n_0 + \log|A_1| + \log|B_1| \cdots + \log|B_{p-1}| + (p+1) \\
&= \log n_0 \cdot |A_1| \cdot |B_1| \cdots |B_{p-1}| + (p+1) \\
&\leq \log|A_1| + \log N_{p-1} + (p+1) \quad (\textit{by Lemma } 6.2.2) \\
&\leq 2\log N + 3 \quad (\textit{by Lemma } 6.2.3)
\end{aligned}
$$

Therefore, the makespan of the schedule is at most $2\log N + pC + 3$. □

We prove that pC is a lower bound for the optimal solution.

Lemma 6.5.2 *If we assume that local transfers take zero unit of time, the makespan is at least pC when A_p is one of the last clusters that receive the message. If there are some clusters that receive the message c' time units later than A_p then the makespan is at least $pC + c'$.*

Proof In the case when there are other clusters that receive the message later than A_p, let A_{p+1}

denote the biggest cluster that receives the message last, and it receives the message c' time units later than A_p ($c' < C$ since otherwise we would have another global phase). We need additional $c' + \lceil \log |A_{p+1}| \rceil$ time units (it is less than $c' + \lceil \log |B_p| \rceil$). $\square$

We thus conclude:

Theorem 6.5.3 *The makespan of the schedule generated by* LCF *is at most 3 times the optimal (with an additive term of 3) when A_p is one of the last clusters that receive the message.*

We now deal with the case when there are other clusters that receive the message later than A_p. Let A_{p+1} denote the biggest cluster that receives the message last, and it receives the message c' time units later than A_p ($c' < C$ since otherwise we would have another global phase). We need additional $c' + \lceil \log |A_{p+1}| \rceil$ time units (it is less than $c' + \lceil \log |B_p| \rceil$).

Lemma 6.5.4 $n_0 \cdot |B_1| \cdots |B_p| \leq N_p$.

Lemma 6.5.5 *$pC + c'$ is a lower bound on the optimal solution.*

Proof Suppose that local transfers take zero unit of time. Then in *LCF*, A_{p+1} receives the message c' time units later than A_p (since the schedule can be obtained by ignoring local phases). Therefore $pC + c'$ is a lower bound on the optimal solution. $\square$

Theorem 6.5.6 *The makespan of the schedule generated by* LCF *is at most 3 times the optimal (with additive term of 4).*

6.5.2 Interleaved LCF

We present another algorithm called *Interleaved LCF*. We show that it is 2-approximation among schedules that use the minimum number of global transfers.

Algorithm *Interleaved LCF*

At every two rounds, a processor that has the message alternately performs the following two steps.

1. Local transfer: if there is any processor in the same cluster that has not received the message, then send the message to it.
2. Global transfer: if there is any cluster that has not received the message, choose the biggest cluster among them and send the message to a processor in the cluster.

We only consider a set of schedules (denoted as S) that minimize the total number of global transfers. Note that schedules in S have the property that each cluster receives only one message from outside ($k-1$ in total). Let OPT_S be the minimum makespan among all schedules in S.

Lemma 6.5.7 *There is a schedule in S with makespan OPT_S in which for any pair of clusters K_i, K_j ($n_i > n_j$), K_i receives a message no later than K_j.*

Proof Given a schedule in S with makespan OPT_S, if there is a pair of clusters K_i, K_j ($n_i > n_j$) and K_i receives a message at time t_i and K_j receives a message at time t_j ($t_i > t_j$), then we can modify the schedule so that K_i receives the message no later than K_j without increasing the makespan.

At time t_j, K_i (instead of K_j) receives the message. K_i can do all transfers that K_j does till time t_i. At time t_i, K_j receives a message. Let x_t processors in K_i received the message just after time t in the *original* schedule. Similarly, let y_t processors in K_j received the message just after time t. Then $x_{t_i} = 1$ and $y_{t_i} \leq n_j$. Note that we cannot swap the roles of two clusters just after t_i since K_i has y_{t_i} messages and K_j has only one message. Therefore, K_i should keep performing transfers as if it is K_j for some time. Let t' be the last time when $x_{t'} \leq y_{t'}$. That is, $x_{t'+1} > y_{t'+1}$.

At time $t'+1$ we need to carefully choose which transfers we should do. Note that just before time $t'+1$, K_i has $y_{t'}$ messages and K_j has $x_{t'}$ messages. In K_i we choose $x_{t'+1} - y_{t'}$ processors to make local transfers so that after $t'+1$, K_i has $x_{t'+1}$ copies of message. Since $x_{t'+1} \leq 2x_{t'} \leq 2y_{t'}$, $x_{t'+1} - y_{t'} \leq y_{t'}$ and therefore, we have enough processors to choose. Similarly, in K_j $y_{t'+1} - x_{t'}$ processors do local transfers so that K_j has $y_{t'+1}$ after $t'+1$. The total number of global transfers coming from K_i and K_j in the original schedule is at most $x_{t'} + y_{t'} - (x_{t'+1} - x_{t'}) - (y_{t'+1} - y_{t'}) =$

$x_{t'} + y_{t'} - (y_{t'+1} - x_{t'}) - (x_{t'+1} - y_{t'})$ and this is exactly the number of remaining processors. Therefore, we have the remaining processors enough to make global transfers. After time $t' + 1$, we can do transfers as in the original schedule. □

We can now consider schedules with the property in Lemma 6.5.7 only. Due to the property, a processor knows the receiver to send a message when it performs a global transfer— the largest cluster that has not received any message. The only thing a processor needs to decide at each time is whether it will make a local transfer or global transfer. By performing local and global transfer alternatively, we can bound the makespan by a factor of two.

Theorem 6.5.8 *The makespan of* Interleaved LCF *is at most 2 times OPT_S.*

Proof Given an optimal schedule with the property in Lemma 6.5.7, modify the schedule so that each operation takes 2 units of time. That is, if a processor performs a local transfer then it is idle in the next time slot and if a processor performs a global transfer, it is idle in the previous time slot. The makespan of the modified schedule is at most 2 times the optimal. It is easy to see that the schedule by *Interleaved LCF* should not be worse than the modified schedule since in *Interleaved LCF*, the processors performs local and global transfers alternatively with no idle time. □

6.6 Experiments

One issue with our broadcasting protocol is that it assumes knowledge of the sizes of the clusters. In some applications, the cluster sizes may not be known accurately in advance. What effect can this have on the broadcasting algorithm Largest Cluster First for example? In the simplest model, we study the effect of having inaccurate information regarding the sizes of the clusters.

We run the *LCF* algorithm using the correct cluster sizes; in addition, we run the *LCF* algorithm by basing the order on advertised sizes that may be off by a factor of 2. For example, for each cluster we let the advertised size be fixed, but change the actual size randomly by either doubling it, or halving it. We now run the protocol where the cluster ordering is made by using the advertised size. We found that there is hardly any change in the performance of the broadcasting algorithm.

In the following experiment, we have roughly 2000 clusters and each cluster has advertised size between 1 and 100 nodes. We choose the actual cluster size using a Zipf distribution, i.e., Prob(a cluster having size i) $= \frac{c}{i^{1-\theta}}$, $\forall i = 1, \ldots, 100$ and $0 \leq \theta \leq 1$, where $c = \frac{1}{H_M^{1-\theta}}$, $H_M^{1-\theta} = \sum_{j=1}^{100} \frac{1}{j^{1-\theta}}$, and θ determines the degree of skewness. We assign θ to be 0. Figure 6.4 shows a histogram of the cluster sizes. We also let C be a parameter and vary this from 10 to 1000. This is a reasonable range as we expect the time to send a message across clusters to be within this range, assuming that the time to send a message within a cluster takes unit time. (If it takes a few milliseconds to send a message from one node to another locally, it may take upto a few hundred, or thousand milliseconds to send a message to a node belonging to another cluster.)

(*Actual*) records the broadcast time when the cluster sizes are the same as their advertised sizes. (*Advertised*) records the broadcast time when the cluster sizes are either half or double their advertised sizes (this choice is made independently and randomly for each cluster). We also compare both these methods to the broadcast time if the algorithm were to use a random ordering of the clusters (*Random*). We also illustrate these times and compare them to the best lower bounds from Section 6.2.1.

6.6.1 Results

As shown in Figure 6.5, we found that (*Actual*) performs the best. This is unsurprising because it has completely accurate information. Note that in all the experiments we ran, it performs at most 1.5 times the lower bound, which is smaller than the theoretical bound of 2. (However, we believe that the broadcasting time is a much closer to the optimal solution even though our lower bounds are not strong enough to argue this.) Moreover, (*Advertised*) performs very close to (*Actual*) (it takes only one more round in all instances). This behavior shows that one needs not to have completely accurate information on the size of the clusters for our Largest Cluster First algorithm to perform well. On the other hand, if the algorithm uses a random ordering of the clusters, it

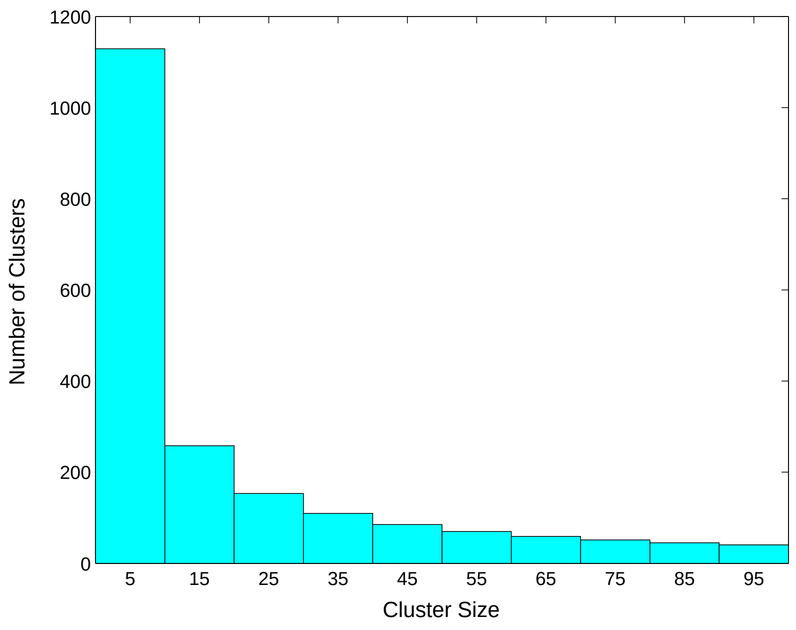


Figure 6.4: A histogram of the cluster sizes.

takes, on average, at least 24% more number of rounds than ($Actual$). In addition, as C increases, ($Actual$) performs closer and closer to the optimal.

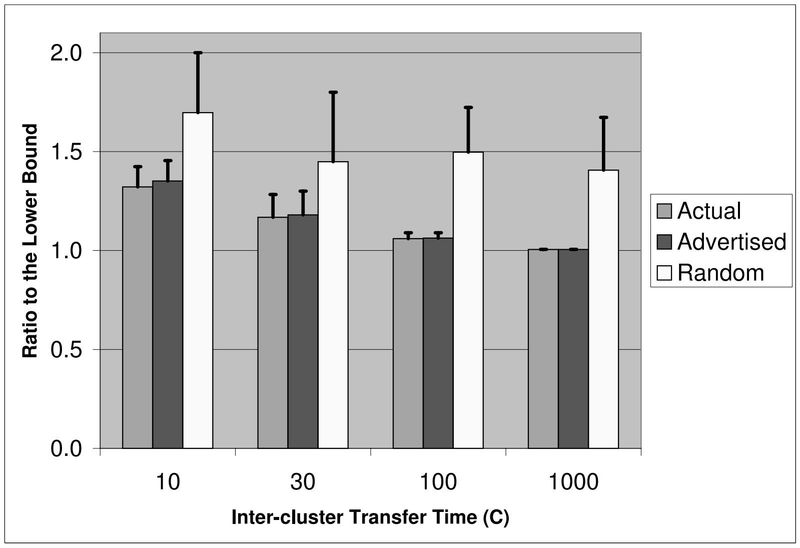


Figure 6.5: The ratio of the number of rounds taken by the algorithms to the lower bound, averaged over 5 inputs, with different values of C (10, 30, 100, and 1000). The maximum ratio appears on the top of each bar.

Chapter 7

Coordinated Data Collection

In this chapter[1], we present our work on a large-scale data collection problem, which arises naturally in the context of wide-area upload applications. The communication model considered in this chapter is more sophisticated than the models considered in previous chapters. We allow different communication capacities between devices and also allow a device to communicate with several other devices simultaneously.

We focus on *application-level* approaches to improving the performance of large-scale data collection (i.e., our approach does not require changes in the routers in the network). We do this in the context of *Bistro* upload framework. We first devise a *coordinated* transfer schedule which would afford maximum possible utilization of available network resources between multiple sources and the destination. We then present a comprehensive study which compares the performance, robustness, and adaptation characteristics of three potential approaches to large-scale data transfers in IP-type networks, namely direct, non-coordinated, and coordinated approaches. We do this using ns2 [50] simulations and within the context of our graph-theoretic model.

7.1 Problem Specification

Our *data collection problem* can be stated as: *Given* a set of source hosts, the amount of data to be collected from each host, a common destination host for the data, and available link capacities between hosts, *our goal is to* construct a data transfer schedule which specifies on which path, in what order, and at what time should each "piece" of data be transferred to the destination host, *where the objective is to* minimize the time it takes to collect all data from the source hosts, i.e., makespan. Since we are focusing on application-level solutions, a path (above) is defined as a sequence of hosts, where the first host on the path is the source of the data, intermediate hosts

[1]This is joint work with C. Chou, W. Cheng, L. Golubchik, and S. Khuller [19].

are other bistros (hosts) in the system, and the last host on the path is the destination host. The transfer of data between any pair of hosts is performed using TCP/IP, i.e., the path the data takes between any pair of hosts is determined by IP routing.

We note that the choice of the makespan metric is dictated by the applications stated above, i.e., there are no clients in the data collection problem and hence metrics that are concerned with interactive response time (such as mean transfer times) are not of as much interest here. Since the above mentioned applications usually process the collected data, the total time it takes to collect it (or some large fraction of it) is of greater significance. Note, however, that our problem formulation (below) is versatile enough that we can optimize for other metrics (if desired), e.g., mean transfer times. We also note that we do not require a distributed algorithm for the above stated problem since Bistro employs a server pull approach, with all information needed to solve the data collection problem available at the destination server. Also not all hosts participating in the data transfer need to be sources of data; this does not change the formulation of our problem since such hosts can simply be treated as sources with zero amount of data to send to the destination.

Moreover, all hosts participating in the data transfer need not be sources of data; this does not change the formulation of our problem since such hosts can simply be treated as sources with zero amount of data to send to the destination.

Although in this work we present our methodology in the context of a single destination, for ease of exposition, we can solve the multi-destination problem as well (by employing either multicommodity flow algorithms [1], or a single commodity min-cost flow algorithm, as in Section 7.3, depending on the requirements). We can also solve the multi-destination problem with multicommodities using multicommodity flow algorithms.

7.2 Overview of Data Collection Approaches

In this section, we give a brief overview of the data collection methods evaluated.

7.2.1 Direct Methods

- *All-at-once.* Data from all source hosts is transferred simultaneously to the destination server.
- *One-by-one.* The destination server *randomly* repeatedly selects one source host from a set of hosts which still have data to send; all data from that source host is then transferred to the destination server.
- *Spread-in-time-GT.* The destination server chooses values for two parameters: (1) group size (G) and (2) time slot length (T). At the beginning of each time slot, the destination server *randomly* selects a group (of size G) and then the data from all source hosts in that group is transferred to the destination server; these transfers continue beyond the time slot length T, if necessary. At the end of a time slot (of length T), the destination server selects another group of size G and the transfer of data from that group begins regardless of whether the data transfers from the previous time slot have completed or not.
- *Concurrent-G.* The destination server chooses a group size (G). It then *randomly* selects G of the source hosts and begins transfer of data from these hosts. The destination server always maintains a constant number, G, of hosts transferring data, i.e., as soon as one of these hosts completes its transfer, the destination server *randomly* selects another source host and its data transfer begins.

Clearly, there are a number of other direct methods that could be constructed as well as variations on the above ones. However, this set is reasonably representative for us to make comparisons (in Section 7.5).

We note, that each of the above methods has its own shortcomings. For instance, if the bottleneck link is not shared by all connections, then direct methods which explore some form of parallelism in data transfer such as the all-at-once method might be able to better utilize existing resources and hence perform better than those that do not exploit parallelism. On the other hand, methods such as all-at-once might result in worse effects on (perhaps already poor) congestion conditions. Methods such as concurrent and spread-in-time require proper choices of parameters and their performance is sensitive to these choices.

Regardless of the specifics of a direct method, due to their direct nature, none of them are able to take advantage of network resources which are available on routes to the destination server other than the "direct" ones (as dictated by IP). Coordinated methods described below are able to take advantage of such resources and therefore result in significantly better performance, as illustrated in Section 7.5.

7.2.2 Non-coordinated Methods

We consider a non-coordinated approach in which each source host chooses a single application-level ("nearly") best path (in term of available bandwidth) to transfer its data to the destination host. Specifically, this non-coordinated method is similar[2] to the current approaches used in RON [2] and Detour [83]. As suggested in [2, 83], in this non-coordinated approach, we only consider re-routing traffic through at most one intermediate host on the way to the destination host. Moreover, for each source host, we first choose a path, p, with one intermediate host, and consider p as the current "best" path. We then consider another potential path, q, and choose q instead of p, if q's available bandwidth is more than 105% of p's available bandwidth, as in [2]. We continue in this manner, until all paths have been considered.

7.2.3 Our Coordinated Approach

An overview of our approach to the problem stated in Section 7.1 and the subsequent evaluation of that approach is as follows:

- *Step 1.* Construct a graph representation of the hosts and the corresponding communication network. This includes specification of nodes, connectivity between nodes, (estimated) capacities of the corresponding connections, and so on.

- *Step 2.* Generate a time-expanded version of the graph constructed in Step 1.

[2]We are not able to use the exact re-routing algorithm in [2, 83] as the *details* of these algorithms are not stated there.

- *Step 3.* Determine a data transfer schedule on the time-expanded graph by optimizing a given objective function under the appropriate set of constraints. In this chapter, we focus on the objective of minimizing the total amount of time it takes to collect the data from the source hosts, i.e., *makespan*, and we place some constraints on splitting and merging of data chunks.

- *Step 4.* Convert the solution produced in Step 3 under the graph-theoretic formulation to a data transfer schedule for a communication network, taking into consideration the network protocols to be used for the transfers (e.g., TCP/IP). As stated in Section 7.1, this schedule must specify on what path and in what order should each "piece" of data be transferred to the destination host, where a path is defined as a sequence of hosts, with the first host on the path being the source of the data, intermediate hosts on the path being other hosts, and the last host on the path being the destination host.

- *Step 5.* Execute the data transfer schedule produced in Step 4 using ns2 [50] in order to evaluate the "goodness" of this data transfer schedule (i.e., this step is performed to evaluate our approach).

The details of Steps 1 through 3 are given in Section 7.3. The details of Step 4 are given in Section 7.4. Moreover, the details of Step 5 (as well as determination of parameters needed in Step 1) and the corresponding performance evaluation results are given in Section 7.5.

Lastly, we use Figure 7.1 to give a high-level example which illustrates the intuition behind the differences among the three approaches described above. Assume that both source hosts S1 and S2 have 1 unit of data to upload to the destination host D. Normally, this would go through network N1. Let us further assume that there is a bottleneck link somewhere between N1 and D and this link has unit capacity available. Therefore, if D pulls data simultaneously from both S1 and S2. It will take 2 units of time to complete the transfer.

Another host, denoted by K, can transfer data to D at rate twice the unit capacity. However, its connection to hosts S1 and S2 is limited at unit capacity. If both S1 and S2 choose to use K to reroute the data to D, it will take 2 units of time to transfer data to K and another unit of

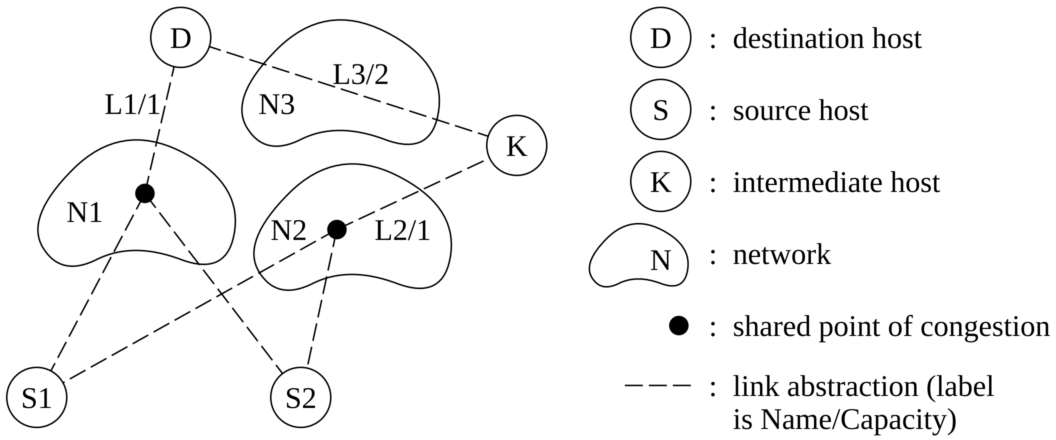


Figure 7.1: A high-level example.

time to transfer data from K to D. The total transfer time is 3 in this case. Chances are that uncoordinated approaches will not choose this solution.

However, if we *coordinate* the data transfers in the following fashion, we can cut the total data transfer time to 1.5 units of time. During the first unit of time, S1 transfers its data directly to D and, *simultaneously*, S2 transfers its data to K. At the end of this time period, there is 1 unit of data left at K. It would take another 0.5 units of time to complete the data transfer[3]. As far as the makespan metrics are concerned, the data transfer from S2 to K is *free* because it occurs *in parallel* with the data transfer from S1 to D. It does not *compete* with the other data transfer. This parallelism cannot be exploited explicitly with other approaches. We believe that in a large network, there is a lot of such parallelism which can be exploited using our approach.

7.3 Graph Theoretic Formulation

In general, the network topology can be described by a graph $G_N = (V_N, E_N)$, with two types of nodes, namely end-hosts and routers, and a capacity function c which specifies the available capacity on the links in the network. The sources $S_1, \ldots S_k$ and the destination host D are a subset of the end-hosts. The example of Figure 7.2(a) illustrates the potential benefits of a

[3]We note that our graph-theoretic coordinated data transfer algorithm given below is able to construct a solution which reduces the total transfer time in the above example to 1.2 time units.

coordinated approach. Here, some chunk of data can be transferred between S_3 and D, while another is transferred, in parallel, between S_2 and S_1 (as staging for the final transfer from S_1 to D). These two transfers would not interfere with each other while attempting to reduce the makespan metric by utilizing different resources in the network in parallel.

Note that, our algorithm (below) *does not know either the topology of the network or the capacity function.* In addition, background traffic exists, which affects the available bandwidth on the links. Hence, we model the network by an *overlay* graph consisting of the set of source hosts and a destination host. (For ease of presentation below we discuss our methodology in the context of source hosts and destination host; however, *any end-host* can be part of the overlay graph, if it is participating in the Bistro architecture. In that case, the node corresponding to this host would simply have zero amount of data to send in the exposition below.) We refer to the overlay graph as $G_H = (V_H, E_H)$. The overlay graph is a directed (complete) graph where $V_H = \{S_1, \ldots, S_k\} \cup \{D\}$. (See Figure 7.2(b) for an example corresponding to Figure 7.2(a); outgoing edges from D are not shown since they are never used.) The capacity function in G_H models available capacity c' on each edge and is assigned as the bandwidth that is available for data transfer between end-hosts. (This takes into account the background traffic, but not any traffic that we are injecting into the network for the movement of data from the sources to the destination.) In other words, this is the bandwidth that is available to us on the path which the *network provides us* in the graph G_N, subject to the background traffic. Since we may not know the underlying topology or the routes that the paths take, we may not be able to properly model conflicts between flows. In other words, node S_2 may not simultaneously be able to send data at rate 1 to each of D and S_3 since the paths that are provided by the network share a congested link and compete for bandwidth. Such knowledge (if available) could be used to specify a capacity function on *sets* of edges, and one could use Linear Programming [25] to obtain an optimal flow under those constraints.

From the overlay graph G_H we construct a "time-expanded" graph $G_{T'}$ [32, 46] (see Figure 7.3) which is the graph that our algorithm will use for computing a schedule to route the data from the sources to the destination[4]. Given a non-negative integer T', we construct this graph as

[4]For clarity of presentation, we omit edges in Figure 7.3 which are not useful for the corresponding data transfer

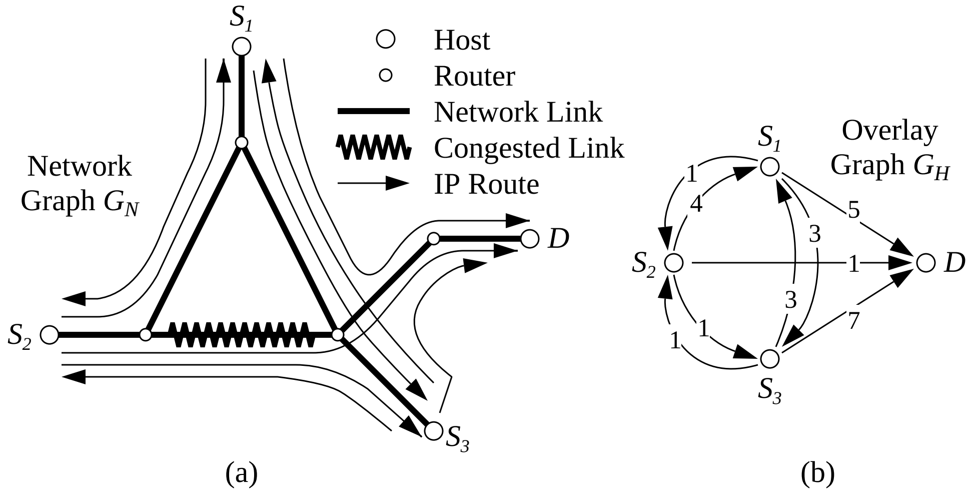


Figure 7.2: Network topology and a corresponding overlay graph.

follows: for each node u in G_H we create a set of $T' + 1$ vertices $u(i)$ for $i = 0 \ldots T'$ and a virtual destination D' in $G_{T'}$. We pick a unit of time t (refer to Section 7.5 for the choice of t) and for each edge (u, v) in G_H, add, for all i, edges in $G_{T'}$ from $u(i)$ to $v(i + 1)$ with capacity $t \cdot c'(u, v)$. (For example, suppose we have available capacity from u and v of 20 Kbps and define t to be 2 seconds. Then, we can transfer 40 Kb of data from u to v in "one unit of time".) Thus, the capacity of the edge from $u(i)$ to $v(i+1)$ models the amount of data that can be transferred from u to v in one unit of time. We will discuss the discrepancies between the model and a TCP/IP network in Section 7.4. In addition, we have edges from $D(i)$ to the virtual destination D', and edges from $u(0)$ to $u(i)$ which are referred to as the "holdover" edges. The latter just corresponds to keeping the data at that node without sending it anywhere.

We first define the min-cost flow problem [1]: given a graph in which each edge has a capacity and unit transportation cost, some vertices are supply nodes supplying flows, some are demand nodes demanding flows, while the total supply equals to the total demand. We want to satisfy all demands, by flows from the supply nodes with the minimum total cost. We use Goldberg's code [37, 38] to find an optimal flow efficiently. We now define a min-cost flow instance on $G_{T'}$: let the supply of $S_i(0)$ be the amount of data to be collected from the source host S_i, and the demand

example.

of D' be the total supply. Figure 7.2(b) shows 12, 15, and 14 units of data have to be collected from S_1, S_2, and S_3, respectively, and the total demand of D' is 41 units. We define the cost of each edge later. Note that by *disallowing* edges from $u(i)$ to $u(i+1)$ for $i > 0$, we hold flow at the source nodes until it is ready to be shipped. In other words, flow is sent from $S_2(0)$ to $S_2(1)$ and then to $S_1(2)$, rather than from $S_2(0)$ to $S_1(1)$ to $S_1(2)$ (which is not allowed since there is no edge from $S_1(1)$ to $S_1(2)$). This has the advantage that the storage required at the intermediate nodes is lower. Hoppe and Tardos [47] argue that allowing edges of the form $u(i)$ to $u(i+1)$ does not decrease the minimum value of t (i.e., makespan).

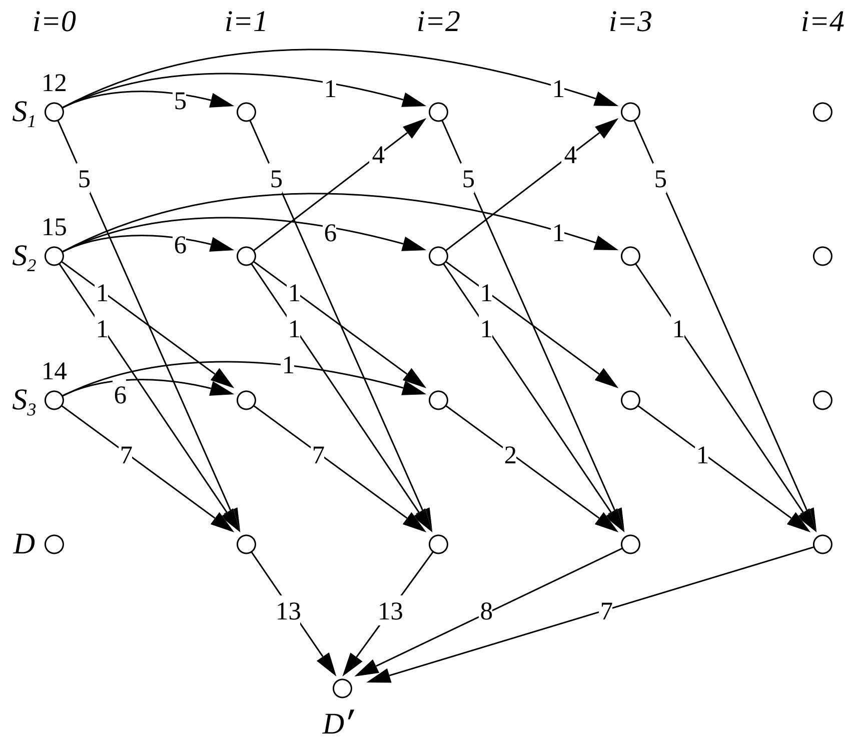


Figure 7.3: Time-expanded graph G_T with $T = 4$.

Our first goal then is to compute the minimum value of T' such that we can route all the data to the destination D' in $G_{T'}$, because, with respect to our flow abstraction, the makespan of our algorithm is T'. Suppose the minimum value of T' is T, we can find it in $O(\log T)$ time by doing a doubling search, followed by a binary search once we find an interval that contains the minimum T for which a feasible solution exists. In other words, we test sequentially if a feasible flow exists in $G_1, G_2, G_4, ...$ until we find a minimum T^* such that a feasible flow exists in G_{T^*} but not in $G_{T^*/2}$. Then we can obtain T by a binary search in the range $T^*/2+1$ and T^*.

Once we know T, we apply a min-cost flow algorithm in the time-expanded graph G_T to obtain an *optimal* flow solution f_T, e.g., the number on the edge in Figure 7.3 corresponds to how much data will be sent by that link at that time, as follows. First note that there could be several feasible flows in G_T that can route all the data to D'. Imposing unit transportation costs on edges guides the algorithm to obtain a feasible flow with certain desirable properties. We associate a cost of $C_1 - C_2 \cdot c'(u, v)$ with every transfer edge $u(i)$ to $v(i+1)$, where C_1 and C_2 are constants and $C_1 \gg C_2 \gg 1$. Thus, our solution would prefer sending data over high capacity links if two solutions have the same total number of transfers. This also provides a more regular pattern in the flow solution. (which can be useful in the PathMerge algorithm described in Section 7.4). To every holdover edge $u(0)$ to $u(i)$, we assign a cost of i. This ensures that data is sent as soon as possible. In other words, among all feasible flows, we prefer the ones with the property that the data arrives earlier at D'. Lastly, the cost is 0 for all other edges. Modifications to this cost function can be made if other properties are desired (e.g., based on network and/or protocol characteristics).

The min-cost flow algorithm runs in $O(n^2 m \log nC)$, where n, m, and C are the number of vertices, the number of edges, and the maximum capacity of a link, in the network flow graph, respectively. If we have b bistros, then $n = (T+1)b$, $m = (T+1)b(b-1)/2$, and $C = ($*maximum amount of data that can be sent in one time unit*$)$ / $($*data unit size*$)$. Thus, the total running time (in worst case) of our algorithm is $O(T^3 b^4 (\log TbC)(\log T))$. In our experiments, the entire computation takes on the order of a few seconds on a Pentium III 650 MHz, and typical values of T, b, and C are 150, 8, and 30, respectively. Moreover, in our problem T is not large so it is feasible to build the entire time-expanded graph and run a min-cost flow algorithm on it. Otherwise, one could use other algorithms (see Hoppe and Tardos [46, 47]) which run in polynomial time rather than pseudo-polynomial time.

Note that in our formulation, we compute the capacity function once initially (refer to Section 7.5), to estimate the available capacity between pairs of hosts. We then assume this to be the available bandwidth for the entire duration of the transfer. Of course, if the transfer is going to take a long time, we cannot assume that the network conditions are static. In this case, we can always compute a new estimate of available bandwidth during the scheduling of the transfer and compute a new transfer schedule for the remaining data. (Our algorithm itself is very fast, and so this does not cause a problem even if the current transfer is stopped, and the schedule is changed.) In fact, the algorithm itself can detect when transfer times are not behaving as predicted and compute a new estimate of capacities. Therefore, we also perform an adaptation study to changing network conditions in Section 7.5. Also note that our algorithm is not complicated to implement since we are working at the application layer, where we only need to control the route within the *overlay* network without any changes to the network protocols (such as IP or TCP). We also note that for ease of exposition, we do not explicitly include I/O bandwidth constraints in our formulation; however, this can easily be included in capacity constraints within the same graph-theoretic formulation. We do not include this in our experiments (below) as currently, I/O bandwidth is not the bottleneck resource in our application. Lastly, the formulation above is quite robust and we can use it to model situations where data may be available at different sources at different times.

Remark: An alternative approach might be to use the overlay graph, G_H, to compute the "best" path in G_H from each host to the destination, independently, e.g., S_2 may choose the path (S_2, S_1, D) since it is the maximum capacity path to D, and send all of its data along this path. This would correspond to the *non-coordinated* approach, and hence, our *coordinated* approach formulation includes the non-coordinated approach as a special case. However, this option does *not* permit for (a) any coordination between transfers from different source hosts, (b) explicit load balancing, as each node makes an independent decision as to which route to send the data on, and (c) maximum possible utilization of available network resources between a source and the destination. More formally, in our time-expanded graph, the non-coordinated method corresponds to a feasible flow in a graph G_{T_i} for some T_i. Note that $T_i \geq T$ where T is the minimum value obtained by our algorithm, which allows for sending of data along multiple paths between a source and the destination. In fact, by sending the data along several paths, our algorithm obtains *a better solution than the non-coordinated method.* This difference becomes especially significant, if several good application level routes exist, but non-coordinated methods send their data along the "best" path, thus causing congestion along this path. In this chapter, we show that the coordinated approach performance significantly better (refer to Section 7.5).

7.4 Transfer Schedule Construction

What remains is to construct a data transfer schedule, f_N (defined as the goal of our data collection problem in Section 7.1), from the flow function f_T computed in Section 7.3, while taking into consideration characteristics of wide-area networks such as the TCP/IP protocol used to transfer the data. This conversion is non-trivial partly due to the discrepancies between the graph-theoretic abstraction used in Section 7.3 and the way a TCP/IP network works. (Below we assume that each data transfer is done using a TCP connection.)

One such discrepancy is the lack of variance in data transfers in the graph-theoretic formulation, i.e., a transfer of X units of data always takes a fixed amount of time over a particular link. This is not the case for data transferred over TCP in a wide-area network, partly due to congestion characteristics at the time of transfer and partly due to TCP's congestion avoidance mechanisms (e.g., decrease in sending rate when losses are encountered). Another discrepancy in the graph theoretic formulation is that it does not matter (from the solution's point of view) whether the X units are transferred as a single flow, or as multiple flows in parallel, or as multiple flows in sequence. However, all these factors affect the makespan metric when transferring data over TCP/IP. Again, these distinctions are partly due to TCP's congestion avoidance mechanisms.

Thus, we believe that the following factors should be considered in constructing f_N, given f_T: (a) size of each transfer, (b) parallelism in flows between a pair of hosts, (c) data split and merge constraints, and (d) synchronization of flows. In this chapter, we propose several different techniques for constructing f_N from f_T, which differ in how they address issues (a) and (d). We first give a more detailed explanation of these issues and then describe our techniques. Note that, we use the term “transfer” to mean the data transferred between two hosts during a single TCP connection.

Size of each transfer.

If the size of each transfer is “too large” we could unnecessarily increase makespan due to lack of pipelining in transferring the data along the path from source to destination (in other words, increased delay in each stage of application-level routing). For example, suppose f_T dictates a transfer of 100 units of data from node S_2 to S_3 to D. S_3 does not start sending data to D until all 100 units of data from S_2 have arrived. If the size of each transfer is 10 units, S_3 can start sending some data to D after the first 10 units of data have arrived. On the other hand, if the size of each data transfer is “too small” then the overheads of establishing a connection and the time spent in TCP’s slow start could contribute significantly to makespan.

In this work, we address the “too small” problem as follows: we ensure that each transfer is of a reasonably large size by carefully picking the time unit and data unit size parameters in the graph construction step (refer to Section 7.5 for details). Second, we can provide a mechanism for merging data transfers which are deemed “too small” (we omit this approach in the interests of brevity; please refer to [20]). The “too large” problem is addressed by a proper choice of the time unit parameter (see Section 7.5).

Parallelism between flows.

One could try to obtain a greater share of a bottleneck link for an application by transferring its data, between a pair of hosts, over multiple parallel TCP connections. However, we do not explore this option here, mainly because it is not as useful (based on our simulation experiments) in illustrating the *difference* between the data collection methods since all these methods can benefit from this. In fact, we made a comparison between a direct method employing parallel connections and our coordinated methods *without* parallel connections, and the coordinated methods could still achieve better performance.

Data split and merge constraints.

The f_T solution allows for arbitrary (although discrete) splitting and merging of data being transferred. However, in a real implementation, such splitting and merging (of data which represents uploads coming from many different clients) can be costly. For instance, in the income tax submission forms example, if we were to arbitrarily split a user's income tax forms along the data transfer path, we would need to include some meta-data which would allow piecing it back together at the destination server. Since there is a cost associated with splitting and merging of data, we allow it only at the source of that data and the destination. To ensure this constraint is met, the first step in our f_N construction techniques is to decompose f_T into flow paths (see details below).

Evaluation of potential additional benefits of splitting and merging is ongoing work. For instance, if we do not want to allow any splitting of the data, we could consider formulating the problem as an unsplittable flow problem. Unfortunately, unsplittable flow problems are NP-complete [65]. Good heuristics for these have been developed recently, and could be used [26].

Synchronization of flows.

The f_T solution essentially synchronizes all the data transfers on a per time step basis, which leads to proper utilization of link capacities. This synchronization comes for free given our graph-theoretic formulation of the data collection problem. However, in a real network, such synchronization will not occur naturally. In general, we could implement some form of synchronization in data transfers at the cost of additional, out-of-band, messages between bistros. Since the Bistro architecture employs a server pull of the data, this is a reasonable approach, assuming that some form of synchronization is beneficial. Thus, we explore the benefits of synchronization.

Splitting the flow into paths.

Given that splitting and merging of data is restricted, we now give details of decomposing f_T into paths, which is the first step in constructing f_N from f_T. To obtain a path from f_T, we traverse the time-expanded graph (based on f_T) and construct a path from the nodes we encounter during the traversal as follows. We start from a source host which has the smallest index number. Consider now all hosts that receive non-zero flows from it. Among those we then choose the one with the smallest index number, and then proceed to consider all hosts that receive non-zero flows from it. We continue in this manner until the virtual destination is reached. The data transferred over the resulting path p is the maximum amount of data that can be sent through p (i.e., the minimum of flow volume over all edges of p). We note that a path specifies how a fixed amount of data is transferred from a source to the destination. For example (in Figure 7.3), a path can be specified as $(S_2(0), S_2(1), S_1(2), D(3), D')$, which says that a fixed amount of data is transferred from node S_2 to node S_1 at time 1, and then from node S_1 to the destination D at time 2 (and D' is the virtual destination). In fact, for this path the value of the flow is 4.

To split the flow network into paths, we first obtain a path using the procedure described above. We then subtract this path from f_T. We then obtain another path from what remains of f_T and continue in this manner until there are no more flows left in f_T. At the end of this procedure, we have decomposed f_T into a collection of paths. (An example of this flow decomposition is given under the description of the PathSync algorithm below and in Figure 7.4.)

Imposing Synchronization Constraints.

What remains now is to construct a schedule for transferring the appropriate amounts of data along each path. We propose the following methods for constructing this schedule which differ in how they attempt to preserve the time synchronization information produced by the time-expanded graph solution.

The PathSync Method.

In this method we employ complete synchronization as prescribed by the time-expanded graph solution obtained in Section 7.3. That is, we first begin all the data transfers which are supposed to start at time step 0. We wait for all transfers belonging to time step 0 to complete before beginning any of the transfers belonging to time step 1, and so on. We continue in this manner until all data transfers in the last time step are complete. We term this approach *PathSync100* (meaning that it attempts 100% synchronization as dictated by f_T).

Recall that the capacity of an edge in the time-expanded graph is the volume of data that can be sent over it during one time unit. Since estimates of available capacity may not be accurate (refer to Section 7.5), and since we may not know which transfers do or do not share the same bottleneck link (unless, e.g., we employ techniques in [82]), it is possible, that some transfers may take a significantly longer time to finish than dictated by f_T. Given the strict synchronization rules above, one or two slow transfers could greatly affect makespan. An alternative is to synchronize only $X\%$ of the transfers. That is, as long as a certain percentage of the data transfers have completed, we can begin all the transfers corresponding to the next time step, except, of course, those that are waiting for the previous hop on the same path to complete. We term this alternative *PathSyncX* where X indicates the percentage of transfers needed to satisfy the synchronization constraints.

An example of PathSync is depicted in Figure 7.4 which shows a collection of paths obtained from decomposing f_T. At time step 0, PathSync100 starts the transfer from $S_1(0)$ to $D(1)$, $S_2(0)$ to $S_3(1)$, $S_2(0)$ to $D(1)$, and $S_3(0)$ to $D(1)$, since all these transfers belong to time step 0. When all these transfers have finished, PathSync100 starts the transfers belonging to time step 1, namely $S_1(1)$ to $D(2)$, $S_2(1)$ to $S_1(2)$, $S_2(1)$ to $S_3(2)$, etc.

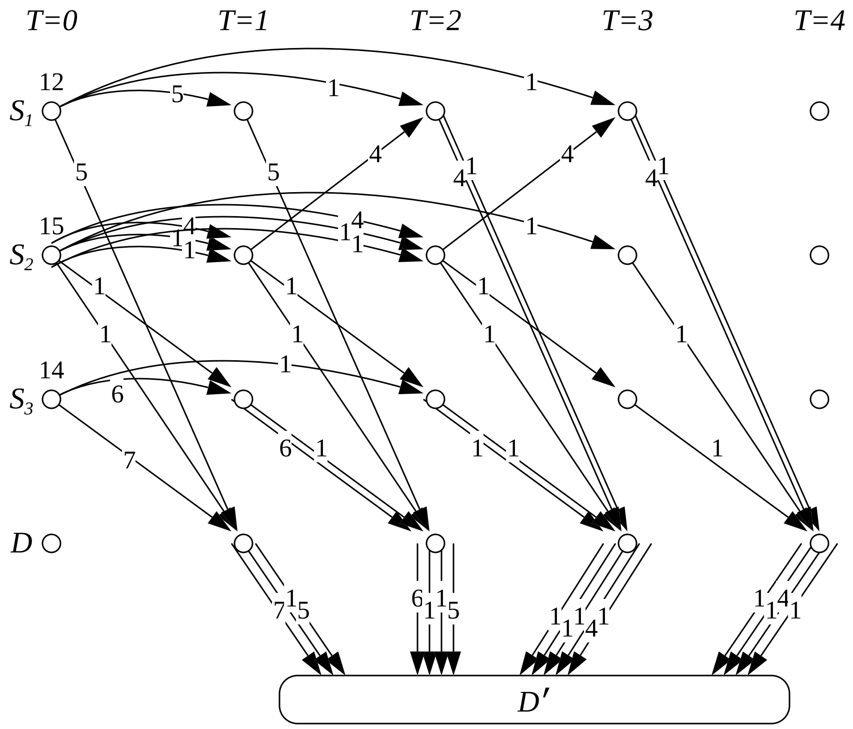


Figure 7.4: Solution obtained after flow decomposition.

The PathSync method performs quite well (refer to Section 7.5), especially when the percentage of transfers that satisfy the synchronization requirements is a bit lower than 100%. This is an indication that it is worth while to attempt to preserve the timing constraints prescribed by the solution of the time-expanded graph (as long as these benefits are not subsumed by the harmful effects of potentially high variance in the transfers). Since synchronization between hosts is not free in a real implementation, we also consider a method which does not require it.

The PathDelay Method.

In the *PathDelay* method we do *not* attempt synchronization between transfers once a transfer along a particular path begins. That is, as long as a particular data transfer along one hop of a path completes, the transfer of that data begins along the next hop of that path. The only synchronization performed in this method is to delay the transfer of that data from the *source* node until an appropriate time, as dictated by f_T, i.e., no inter-host synchronization is needed. For example, after the decomposition of f_T into paths, there is a path $(S_2(0), S_2(2), S_1(3), D(4), D')$ of size 4 (see Figure 7.4). Since the data is held at the source S_2 until time step 2 in f_T, we schedule the $S_2(2)$ to $S_1(3)$ transfer at "real" time $2 \cdot t$, where t is our time unit (refer to Section 7.5).

One could also create variations on PathDelay by expanding or contracting the time unit, used in computing f_T, when constructing f_N, again to account for variance in data transfer in a real network as compared to the graph-theoretic formulation. For instance, *PathDelayX* would refer to a variation where the time unit t in f_T is modified to be Xt in f_N.

7.5 Performance Evaluation

In this section we evaluate the performance of the various data transfer methods and illustrate the benefits of using a *coordinated* approach. This evaluation is done through simulation; all results are given with at least $95\% \pm 5\%$ confidence.

Simulation Setup

For all simulation results reported below, we use ns2 [50] in conjunction with the GT-ITM topology generator [49] to generate a transit-stub type graph with 152 nodes for our network topology. The number of transit domains is 2, where each transit domain has, on the average, 4 transit nodes with there being an edge between each pair of nodes with probability of 0.6. Each node in a transit domain has, on the average, 3 stub domains connected to it; there are no additional transit-stub edges and no additional stub-stub edges. Each stub domain has, on the average, 6 nodes with there being an edge between every pair of nodes with probability of 0.2. A subset of our simulation topology (i.e., without stub domain details) is shown in Figure 7.5. The capacity of a "transit node

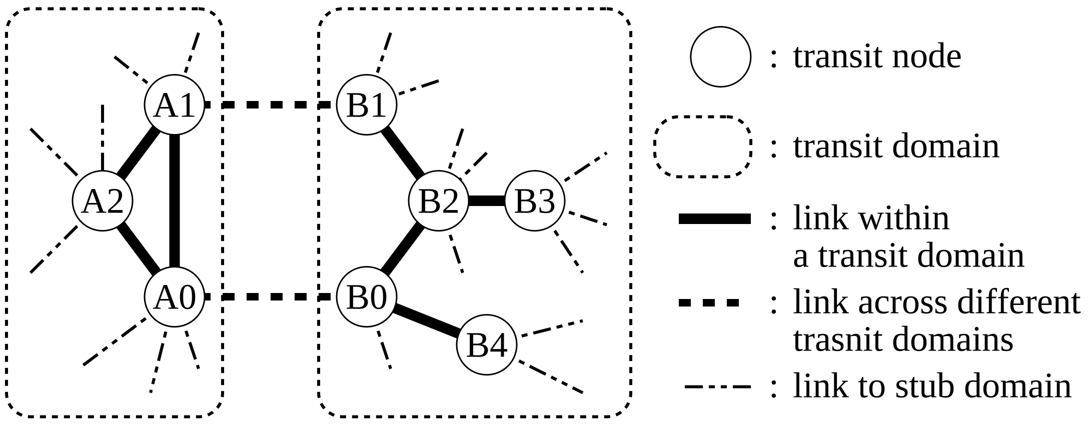

A1
A2
A0
B1
B2
B3
B0
B4
: transit node
: transit domain
: link within a transit domain
: link across different trasnit domains
: link to stub domain

Figure 7.5: The simulation topology.

to transit node" edge within the same transit domain is 10 Mbps. The capacity of a "transit node to transit node" edge across different transit domains is 5 Mbps. The capacity for a "transit node to stub node" edge or a "stub node to stub node" edge is 2.5 Mbps. Our motivation for assigning a lower capacity to the "transit node to transit node" edge across different transit domains is to emulate poorer performance conditions that exist at the peering points [71]. Moreover, we linearly scale the propagation delay between nodes generated by the GT-ITM topology generator [49] such that the maximum round trip propagation delay in our topology is 80 ms. Note that, the size and parameters of our network model and the simulation setup are motivated by what is practical to simulate with ns2 in a reasonable amount of time. However, since our goal is a relative comparison of the methods, this will suffice.

We locate the destination server in the stub domain connected to $A1$, and we locate 7 other bistros in stub domains connected to other transit nodes. Each bistro holds a total amount of data which is uniformly distributed between 25 and 75 MBytes with an additional constraint that the total amount of data in all bistros is 350 MBytes. In addition to the data collection traffic, we setup 0 to 120 background traffic flows from nodes attached to transit domain B to nodes attached to transit domain A. In our experiment, the ratio of the number of background flows in peering point (B0,A0) to the number of background flows in peering point (B1,A1) is 1:3 (asymmetric). We also investigated how the methods behave under different ratios between the peering points such as 1:1 (symmetric), 2:1, and 1:2; the results indicate similar trends. (We do not include them here in the interests of brevity.) The background traffic pattern is similar to that in [82]. Each background flow is an infinite FTP with a probability of 0.75. Otherwise, it is an on-off CBR UDP flow. The average on-time and off-time is chosen uniformly between 0.2 and 3 seconds. The average rate (including the off periods) is chosen so that the expected total volume of UDP traffic through a peering point takes up 5% of the capacity of that point. (Similar trends persist under different volumes of UDP traffic; we do not include details of these results here in the interests of brevity.) To illustrate a reasonably interesting scenario, all nodes participating in background traffic are located in stub domains that are different from those holding the bistros participating in data collection traffic. This choice avoids the non-interesting cases (for makespan) where a single bistro ends up with an extremely poor available bandwidth to *all* other bistros (including the destination server) and hence dominates the makespan results (regardless of the data transfer method used).

Construction of Corresponding Graph

We now give details of constructing graph G_H of Section 7.3 from the above network. The eight bistros make up the nodes of G_H, with the destination bistro being the destination node (D) and the remaining bistros being the source nodes (S_i) with corresponding amounts of data to transfer. The link capacities between any pair of nodes in G_H are determined by estimating the end-to-end mean TCP throughput between the corresponding bistros in the network. In our experiments these throughputs are estimated in a separate simulation run, by measuring the TCP throughput between each pair of bistros separately while sending a 5 MByte file between these bistros. We repeat the measurement 10 times and take the average to get a better estimation. These measurements are performed with background traffic conditions corresponding to a particular experiment of interest but without any data collection traffic or measurement traffic corresponding to other bistro pairs. Although a number of different measurement techniques exist in the literature [18, 27, 53, 68, 75, 85], we use the above one in order to have a reasonably accurate and simple estimate of congestion conditions for purposes of *comparison of data collection methods.* However, we note, that it is *not* our intent to advocate particular measurement and available bandwidth estimation techniques. Rather, we expect that in the future such information will be provided by other Internet services, e.g., such as those proposed in SONAR [78], Internet Distance Map Service (IDMaps) [35], Network Weather Service (NWS) [91], and so on. Since these services will be provided for many applications, we do *not consider bandwidth measurement as an overhead* of *our* application but rather something that can be amortized over many applications.

In order to construct G_T from G_H we need to determine the time unit and the data unit size. The bigger the time unit is, the less costly is the computation of the min-cost flow solution but potentially (a) the less accurate is our abstraction of the network (due to discretization effects) and (b) the higher is the potential for large transfer sizes (which in turn contribute to lack of pipelining effects as discussed in Section 7.4). The smaller the time unit is, the greater is the potential for creating solutions with transfer sizes that are "too small" to be efficient (as discussed in Section 7.4). Similarly, the data unit size should be chosen large enough to avoid creation of small transfer sizes and small enough to avoid significant errors due to discretization (as discussed in Section 7.4).

In the experiments presented here we use a time unit which is 100 times bigger than the maximum propagation delay on the longest path, i.e., 8 sec. (This choice is motivated by the fact that in many cases we were not able to run ns simulations with smaller time units as the resulting number of flows was too large; a smaller time unit did *not* present a problem for our theoretical formulation.) The data unit size is chosen to ensure that the smallest transfer is large enough to get past the slow start phase and reach maximum available bandwidth without congestion conditions. Since without background traffic a bistro can transmit at a maximum window size of 2.5 Mbps $\times$

80 ms (on the longest path), we use a data unit size a bit larger than that, specifically 64 KBytes.

Performance Metrics.

The performance metrics used in the remainder of this section are: (a) makespan, i.e., the time needed to complete transfer of total amount of data from all bistros, (b) maximum storage requirements averaged over all bistros (not including the destination host since it must collect all the data), and (c) mean throughput of background traffic during the data collection process, i.e., we also consider the effect of data collection traffic on other network traffic. We believe that these metrics reflect the quality-of-service characteristics that would be of interest to large-scale data collection applications (refer to Section 7.1).

Evaluation Under the Makespan Metric.

We first evaluate the direct methods described in Section 7.2 using the makespan metric. As illustrated in Figure 7.6(a) direct methods which take advantage of parallelism in data delivery (such as all-at-once) perform better under our simulation setup. Intuitively, this can be explained as follows.

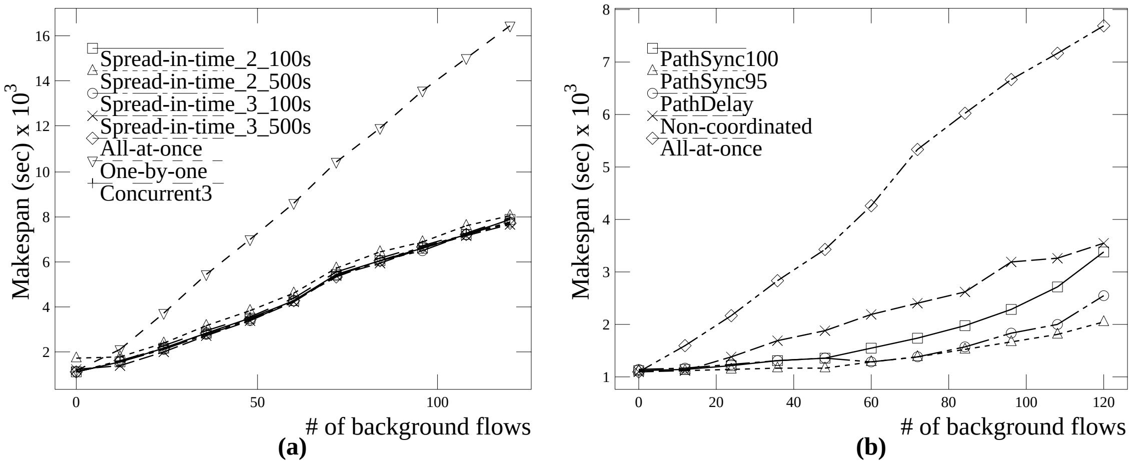


Figure 7.6: Direct, non-coordinated, and coordinated methods under the makespan metric.

Given the makespan metric, the slowest bistro to destination server transfer dominates the makespan metric. Since in our case, the bottleneck which determines the *slowest* transfer in direct methods is not shared by all bistros, it makes intuitive sense to transfer as much data as possible, through bottlenecks which are different from the one used by the slowest transfer, in parallel with the slowest transfer.

Since all-at-once is a simple method and it performs better than or as well as any of the other direct methods described in Section 7.2 under the makespan metric in our experiments, we now compare just the all-at-once method to our coordinated methods. We include non-coordinated methods in this comparison for completeness. This comparison is illustrated in Figure 7.6(b) where we can make the following observations. All schemes give comparable performance when there is no other traffic in the network (this makes intuitive sense since the capacity near the server is the limiting resource in this case). When there is congestion in the network and some bistros have significantly better connections to the destination server than others, our coordinated methods result in a significant improvement in performance, especially as this congestion (due to other traffic in the network) increases. For instance, in Figure 7.6(b), using PathSync95 we observe improvements (compared to direct methods) from 1.9 times under 24 background flows to 3.7 times when the background traffic is sufficiently high (in this case at 120 flows). PathSync95 is 1.7 times better than the non-coordinated method under 120 flows.

As shown in Figure 7.6(b), enforcing full synchronization (as in PathSync100) can be harmful which is not surprising since a single slow stream can lead to (a) significant increases in overall data collection time (although nowhere as significant as the use of direct methods) and (b) increased sensitivity to capacity function estimates and parameter choices in G_H and G_T. We can observe (a), for instance, by comparing the overall performance of PathSync100 and PathSync95 in Figure 7.6(b). Regarding (b), intuitively, overestimating the capacity of a link may result in sending too much data in one time slot in a particular transfer in our schedule, which may delay the whole schedule as we fully synchronize all transfers (in the interests of brevity; please refer to [20]).

We note that if the packet size of the background traffic at the time the capacity estimations were done is different from those at the time the data is collected, PathSync100 performed anywhere from almost identically to two times worse; this is another indication that it is sensitive to capacity function estimates. We also tried modifications to data unit size (during the discretization step in constructing G_H and G_T) and observed similar effects on PathSync100, for reasons similar to those given above. (We do not include details of these results here in the interests of brevity).

Above observations raise another question, which is how much synchronization is really needed in the data collection schedule. By comparing PathDelay with PathSync (and its variants) one might say that ensuring that transfers are initiated at the appropriate times (and then not synchronizing them along the way) is sufficient, since PathDelay performs pretty well in the experiments of Figure 7.6(b). However, the experiments in this figure are relatively small scale and hence have relatively few hops in the paths constructed from f_T. Other experiments indicate that as the number of hops on a path (in G_T) increases, PathDelay begins to suffer from getting out of sync with the schedule computed in f_T and performs much worse than PathSync95, for instance. (We do not include details of these results here in the interests of brevity.)

Remark: One question might be whether the notion of simply assigning time slots (to bistros) during which to transfer data directly to the destination server is a reasonable approach. Since this is essentially the idea behind direct methods such as spread-in-time, and since they performed *significantly* worse than the *coordinated* methods in the experiments illustrated above, we believe that such methods do not lead to sufficiently good solutions.

Evaluation Under the Storage Metric.

Next, we evaluate the different methods with respect to the storage requirements metric. We note that the direct methods do not require additional storage, i.e., beyond what is occupied by the original data itself. In contrast, non-coordinated and coordinated methods do, in general, require additional storage, since each bistro might have to store not only its own data but also the data being re-routed through it to the destination server.

Figure 7.7 illustrates the *normalized* maximum per bistro storage requirements, averaged over all bistros (other than the destination), of the non-coordinated and coordinated methods as a function of increasing congestion conditions. These storage requirements are normalized by those of the direct methods. We use the direct methods as a baseline since they represent the inherent storage requirements of the problem as noted above. As can be seen from this figure, the additional storage requirements of our coordinated algorithms are small. In all experiments

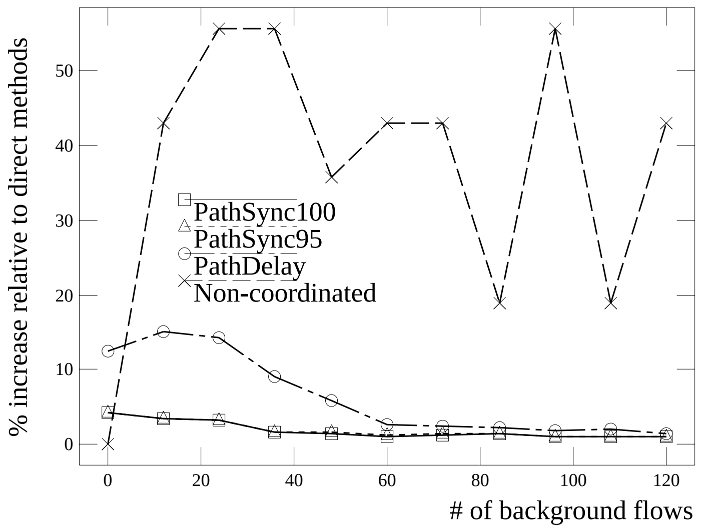


Figure 7.7: Non-coordinated and coordinated methods under the storage metric.

performed by us, storage overheads of all PathSync variations were no more than 5%. PathDelay resulted in storage overheads of no more than 15% (this makes sense since greater storage is needed when less stringent flow synchronization is used). We believe these are reasonable given the above improvements in overall data collection times (and, also given the current storage costs). Note that the storage requirement of the non-coordinated method is high because multiple data flows from different source hosts may be re-routed to the same intermediate host at the same time.

Evaluation Under the Throughput Metric.

We also evaluate the non-coordinated and coordinated methods under the *normalized* mean throughput metric, i.e., their effect on the throughput of the background traffic which represents other traffic in the network. The results are normalized by the throughput achieved by the background ftp traffic *without* presence of the data collection traffic.

We first evaluate the throughput of the direct methods. As illustrated in Figure 7.8(a), the one-by-one method allows for the highest background traffic throughput. This is not surprising, since one-by-one is the most conservative direct method in the sense that it injects the data collection traffic into the network one flow at a time. As can be seen in Figure 7.8(b), the non-coordinated and coordinated methods result in lower background traffic throughput, but not significantly. The largest difference we observed was no more than 16% (for coordinated and non-

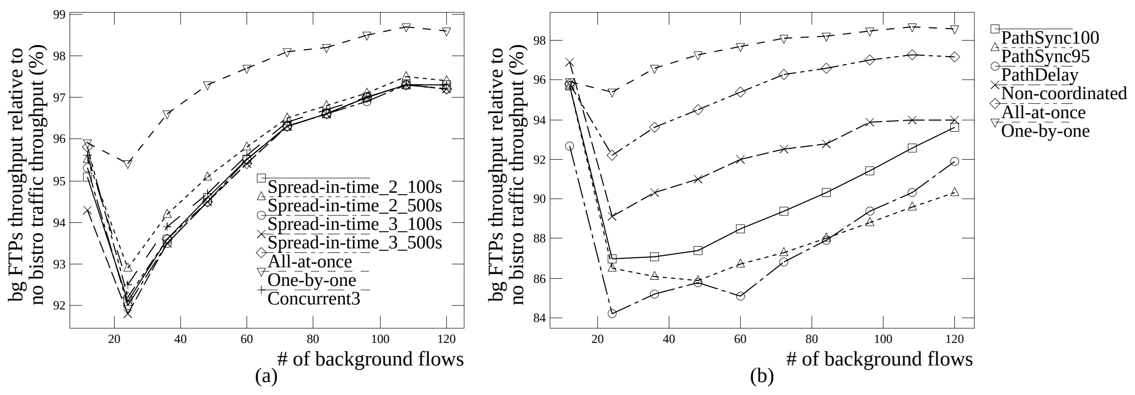


Figure 7.8: Direct, non-coordinated and coordinated methods under the throughput metric.

coordinated methods). This, of course, is not surprising since the coordinated and non-coordinated methods are more aggressive than direct methods in taking advantage of bandwidth available in the network. We believe that this is an indication that they are taking such advantage without significant adverse effects on other traffic in the network.

Remark: Since our extensive evaluation of a variety of direct methods showed that "All-at-once" method performs as well as any of the other direct methods in term of makespan metric, we use that method as a representative "direct method". Similarly, we use the "PathSync95" method as our a representative "coordinated" method.

Effect of UDP traffic

We investigate the effect of UDP traffic on the makespan metric. In Figures 7.9(a) and (b), 7.9(c) and (d), and 7.9(e) and (f) show the makespan metric under 0%, 5%, and 20% of UDP traffic, respectively. Figures 7.9(a), 7.9(c), and 7.9(e) show the symmetric case (1:1) and Figures 7.9(b), 7.9(d), and 7.9(f) show the asymmetric case (2:1). As we can see, Figures 7.9(a), 7.9(c), and 7.9(e) are very similar, so as Figures 7.9(b), 7.9(d), and 7.9(f). Qualitatively, the observations we made above, in comparing direct, non-coordinated, and coordinated methods, still hold under different volumes of UDP traffic, i.e., similar trends persist.

Robustness Study

The motivation for us to do a robustness study of different data collection methods is (i) it is not easy to get a good estimate of available bandwidth of a path and (ii) network congestion conditions might change over time, and hence it is important to see how sensitive the results are to the accuracy of available bandwidth information. To this end we use perturbed values of available bandwidth in computing the data transfer schedule for both non-coordinated and coordinated methods. That is, we deviate available bandwidth values from the actual estimates to emulate inaccuracies in available bandwidth estimates. Note that, the direct method does not use available bandwidth estimation information, and is therefore unaffected by the errors in the estimation.

Since more interesting scenarios correspond to inaccuracies in bandwidth estimation of congested points, (B0,A0) and (B1,A1), we consider the following two cases: (a) increasing the mis-estimates of the available bandwidth of paths passing through link (B0,A0) while decreasing the mis-estimates of the available bandwidth of paths passing through link (B1,A1), and (b) decreasing the mis-estimates of the available bandwidth of paths passing through link (B0,A0) while increasing the mis-estimates of the available bandwidth of paths passing through link (B1,A1). We ran a number of experiments corresponding to different loadings on peering points and different volumes of UDP traffic, and the results were qualitatively similar. Therefore, we only present an experiment with the ratio of traffic loads between peering point (B0,A0) and peering point(B1,A1) being 1:3. Figure 7.10 shows a series of results from over-estimating the available bandwidth of paths passing through the more congested link by 70% in (a), 50% in (b), and 30% in (c), to accurate estimation in (d), to under-estimating the available bandwidth of paths passing through the more congested link by 50% in (e), and 70% in (f). Although the coordinated method is affected by estimation errors in all paths, since it uses multiple paths to perform transfers, the effect is relatively small. For instance, even when the available bandwidth of the paths passing through the more congested link is over-estimated by 70%, as in Figure 7.10(a), the makespan is increased by 46% when comparing to the same method with accurate bandwidth estimation. If we under-estimate the available bandwidth by 70%, as in Figure 7.10(f), the makespan is increased by 24%. However, note that in all cases, the non-coordinated method performs worse than the coordinated method, anywhere from 37% to 142%. Furthermore, for the non-coordinated method, a mis-estimation can be more dangerous; e.g., a high over-estimation of the available bandwidth of a more congested link, as in Figures 7.10(a) and 7.10(b), can result in most of the traffic being routed through the more congested path. Thus, the resulting performance would degrade sharply and can be even worse than that of the direct method. As shown in Figure 7.10(a), the makespan of the non-coordinated method is more than double of the makespan of the same method with accurate bandwidth estimation. On the other hand, since the non-coordinated method chooses the best path (in term of available bandwidth), this over-estimation needs to be sufficiently high in order to force the wrong choice of best path. For instance, in Figure 7.10(c), it does not affect the makespan. We note that, under-estimating the available bandwidth of paths passing through a highly congested link does not affect the non-coordinated method since the method never picks that path.

Adaptation Study

In order to explore the potential for *adaptation* to changes in network conditions of the various data collection methods, we consider a comparison of our coordinated approach with a non-coordinated method in a more dynamic network environment. (We do not include direct methods in this study as it is not possible for them to adapt.) Specifically, by more dynamic network conditions we mean that the changes occur in network conditions, at the session level of the background traffic, while the data collection process is in progress. We note that, although the background traffic used in the above experiments is dynamic at the packet level, it is basically static at the session level, i.e., the number of background traffic flows does not change while the data collection process proceeds. Therefore, in this adaptation study we not only consider the background traffic conditions explored above, but we also allow background traffic flows to join and leave while the data collection process is in progress.

To illustrate a more interesting scenario, we consider the case where changes in the background traffic (i.e., the dynamic network conditions) occur in flows going through links (B0,A0) and (B1,A1). That is, at the beginning of the simulation the ratio of traffic loads between peering point (B0,A0) and peering point (B1,A1) is $1 : r$ with $r > 1$; after some time passes, i.e. , T_{dyn} sec later, the ratio of traffic loads between the above two peering points becomes $r : 1$. In order to give a fair comparison between the non-coordinated and our coordinated methods, both schemes are given the same end-to-end network information and T_{dyn} values. That is, we assume that both methods know exactly when the background traffic conditions change. Given the above information, both the non-coordinated method and our coordinated method use the same approach to adjust their data transfer schedules, which is as follows. At time T_{dyn} both the non-coordinated method and our coordinated method will (i) immediately stop all current data transfers and (ii) re-compute their data transfer schedules (using their respective algorithms) according to the current location and amount of data which has not yet reached the destination host. That is, the system's state (in terms of new network conditions and state of data transfer) at time T_{dyn} is taken as the (new) input to the coordinated and non-coordinated algorithms, and these algorithms are re-run (on these new initial conditions) to produce new data transfer schedules. After this computation is done, both methods will then proceed with the new data transfer schedules until all transfers are completed.

We note that, in the performance study below we do not take into consideration or evaluate, for either method, the various overheads associated with the adaptation process, such as the time needed to coordinate hosts during the transfer schedule modification process. (We also assume that this is done through an out-of-band control channel as in other control processes in Bistro.) A reason for not evaluating such overhead here is that they are largely a function of the specific control protocols used between the hosts participating in the transfers. We do acknowledge that such overheads will likely be different in the coordinated and the non-coordinated methods. And, a reasonable evaluation of these overheads is part of our on-going research efforts.

Of course, in a real system one would also need an approach for detecting or predicting when network conditions have changed or rather when they have changed sufficiently to make the computation and use of a new transfer schedule worth-while. In order to make an appropriate decision of whether or not to adapt a data transfer schedule under these circumstances, we would require an approach for evaluating the associated costs (overheads) and benefits (reduced makespan time) of this adaptation process. These are not simple issues, and they are also part of our ongoing research efforts.

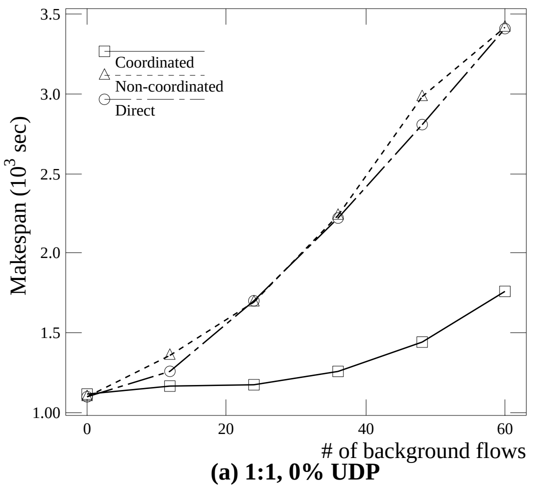
Figure 7.11 depicts results of our adaptation study, where we compare our coordinated method with the non-coordinated method using the makespan metric. In this figure, we use $r = 3$ with 5% background UDP traffic in the peering points, where the number of background FTP flows is indicated by the X-axis. The only difference between Figures 7.11(a) and 7.11(b) is that we use $T_{dyn} = 200$ in Figure 7.11(a), and we use $T_{dyn} = 700$ in Figure 7.11(b). In the results of Figure 7.11 our coordinated method performs better than the non-coordinated method — it shows an up to a 118% improvement in makespan in Figure 7.11(a) and an up to a 182% improvement in makespan in Figure 7.11(b). Moreover, the following interesting observations about the results in Figure 7.11 can be made.

Firstly, in both cases, (a) $T_{dyn} = 200$ and (b) $T_{dyn} = 700$, makespan of the non-coordinated method under 36 background flows is worse than under 60 (or 48) background flows. Intuitively, the makespan metric should increase as the number of background flows increases. This unintuitive result might be explained as follows. By time T_{dyn} when network conditions change, the non-coordinated approach may have already transferred too much data in the "wrong" direction, i.e., to the wrong intermediate host, which may have been a good choice for the network conditions before T_{dyn} but a bad choice for the network conditions after that time. That is, since the non-coordinated method only chooses a *single* "best" path, it is in a sense too aggressive about application-level re-routing of the data around the current network congestion points on that path. And, by the time network conditions change, it is "too committed" to what becomes (after T_{dyn}) a poor application-level path. Hence, it suffers a greater penalty for re-routing data to what becomes (after T_{dyn}) a better path. On the other hand, our coordinated method attempts to utilize all available application-level paths, in an appropriate manner, and hence appears to be more "immune" or more robust to future changes in network conditions. (This is consistent with our robustness study above.) Of course, the model of changes in network conditions used in this study may in a sense be too drastic. However, it is used here in order to simplify the exposition of the differences between the two approaches. A study using a more gradual model of network condition changes is part of our future research efforts.

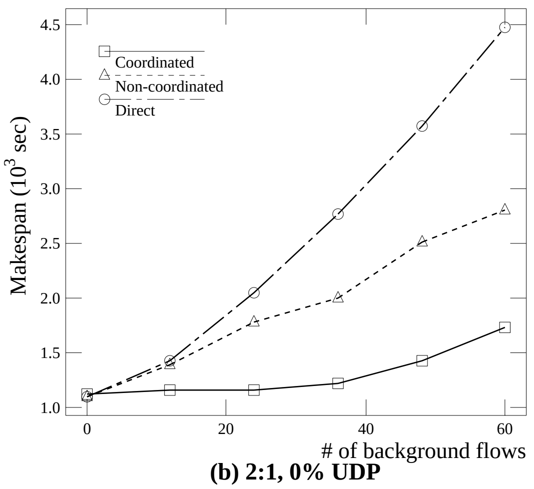
Another interesting observation is that the performance of the non-coordinated method in Figure 7.11(b) is worse than its performance in Figure 7.11(a). Since T_{dyn} is larger in Figure 7.11(b) than in Figure 7.11(a), more data may have ended up being re-routed in this case to the "wrong" intermediate node (as explained above) before changes in network conditions occurred. Of course, further increasing T_{dyn} can improve the performance of the non-coordinated method, as compared to case in Figure 7.11(a), as this would allow time for a greater fraction of the data to reach the final destination before changes in network conditions occur.

Lastly, we note that we performed many more adaptation experiments, using various values for T_{dyn} as well as for the ratio of traffic loads in the peering points. Qualitatively, the results show similar trends as in Figure 7.11. Hence, we do include results of these experiment in the interests of brevity.

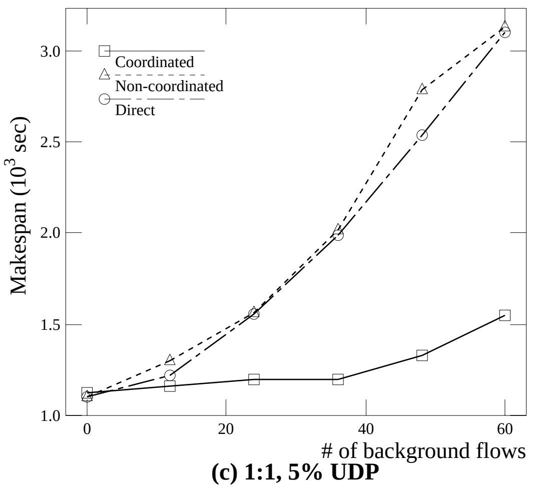
In summary, we believe that the above adaptation study illustrates that our coordinated approach has a greater potential for adaptation than non-coordinated methods. This is largely due to its efforts to utilize all available network resources in constructing a data transfer schedule.



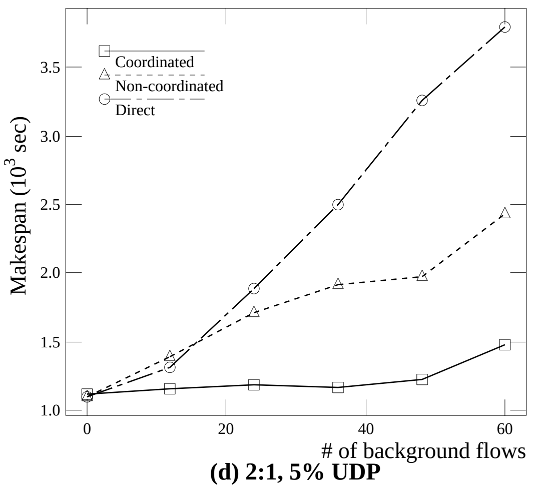
(a) 1:1, 0% UDP



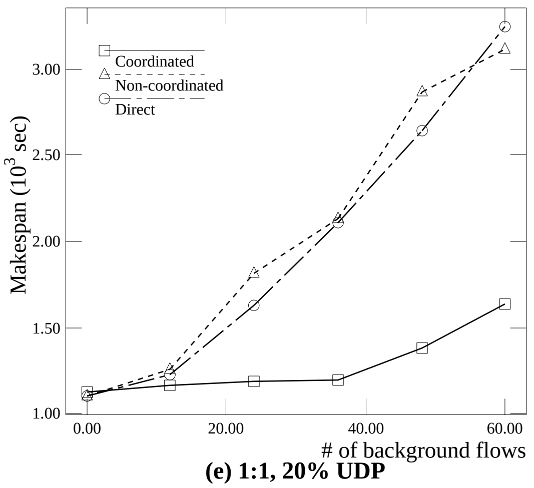
(b) 2:1, 0% UDP



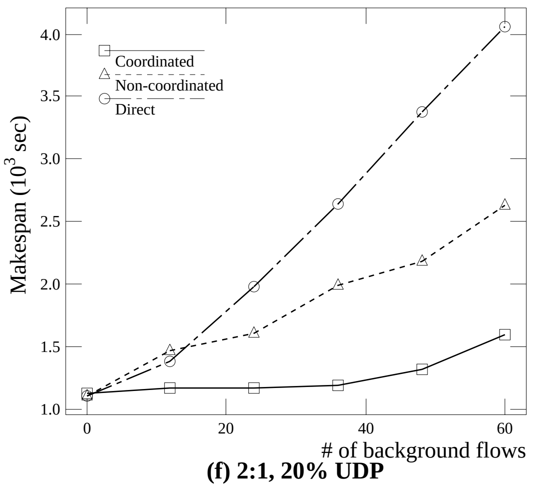
(c) 1:1, 5% UDP



(d) 2:1, 5% UDP

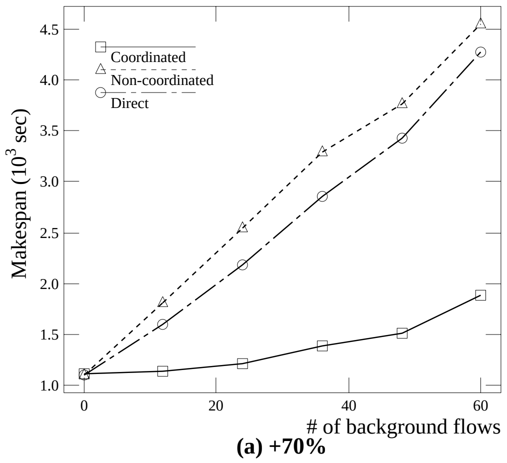


(e) 1:1, 20% UDP

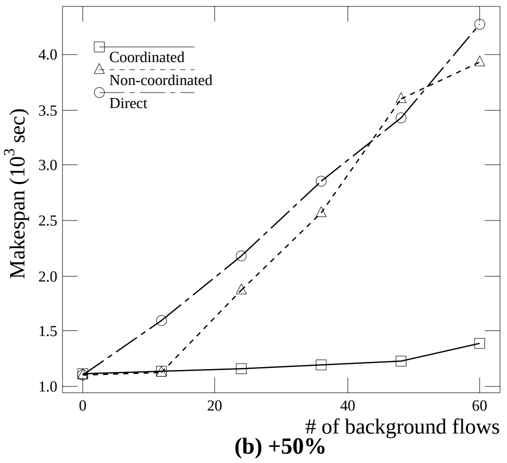


(f) 2:1, 20% UDP

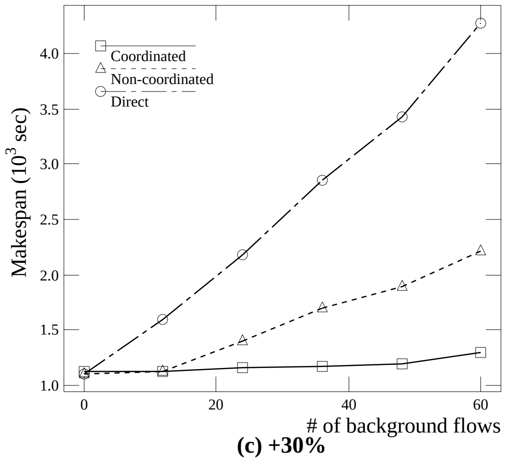
Figure 7.9: Makespan comparison under the effect of UDP traffic, and symmetric (1:1) and asymmetric (2:1) traffic.



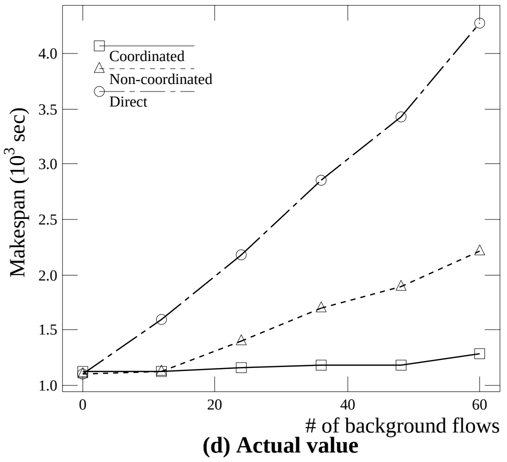
(a) +70%



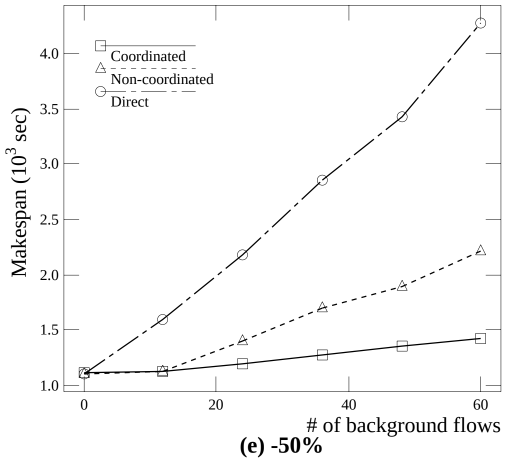
(b) +50%



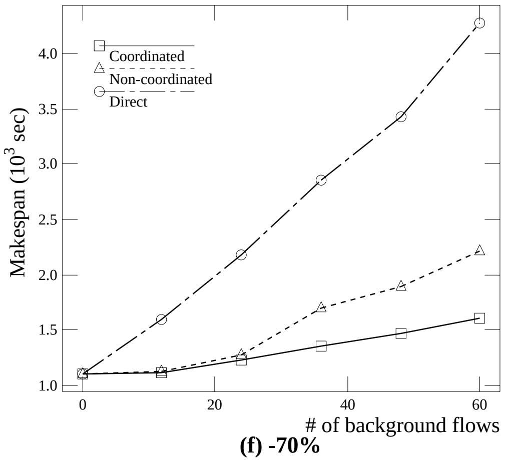
(c) +30%



(d) Actual value



(e) -50%



(f) -70%

Figure 7.10: Sensitivity to bandwidth estimation under 5% UDP traffic and ratio of loads on peering points of 1:3, under the makespan metric. Individual figure captions indicate the percentage of mis-estimation of paths going through (B1,A1).

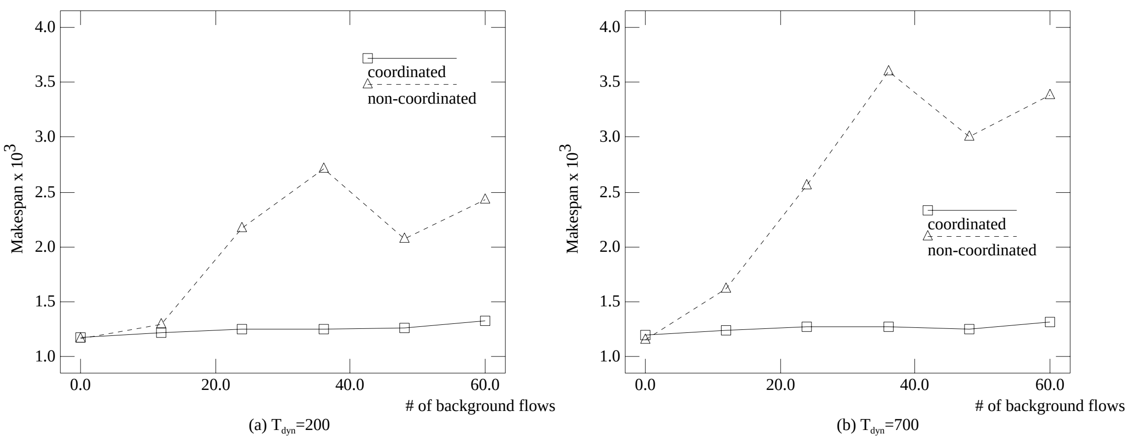


Figure 7.11: Non-coordinated and coordinated methods under dynamic changes in the number of background flows.

Chapter 8

Conclusions and Future Work

In this thesis we studied a number of data dissemination and collection problems that arise when managing large amounts of data over a communication network. Broadcasting and gossiping problems resemble some of the data dissemination problems we considered. However, previous works have mainly concentrated on assuming two parties may exchange all the information they know in constant time, and assuming the underlying communication model is homogeneous. Our work addressed these assumptions, and provides a broadcasting algorithm that is more applicable on a wide-area network. Moreover, to collect data from a set of hosts in a large-scale public network like the Internet, we addressed two key problems: the available bandwidth can fluctuate, and the network may not choose the best route to transfer the data between two hosts.

In this thesis, we studied problems in three main areas: data dissemination problems that generalizes broadcasting and gossiping in a local-area network (Chapters 3 to 5), broadcasting in a wide-area network (Chapter 6), and large-scale data collection (Chapter 7). In the first two parts, we developed approximation algorithms for these problems, and proved one of the problems is NP-hard. In the third part, we proposed a coordinated approach for computing data collection schedules using network flows. We now summarize our contributions for each of the above problems and list several possibility of future work.

In Chapter 3, we considered the single-source multicast problem, where there is one source disk s that has all Δ items and others do not have any item in the beginning, and we would like to send item i to disks in set D_i. We developed an algorithm where D_i can be an arbitrary subset of disks. The number of rounds required by our algorithm is at most $\Delta + OPT$ where OPT is the minimum number of rounds required for this problem. Our algorithm is obviously a 2-approximation for the problem, since Δ is a lower bound on the number of rounds required by the optimal solution.

In Chapter 4, we considered the multi-source broadcast problem and presented an algorithm that takes at most 3 more rounds than the optimal solution.

In Chapter 5, we first presented a polynomial-time 4-approximation algorithm for the multi-source multicast problem. We then showed how to improve it to give a $(3 + o(1))$-approximation algorithm. After that we presented a 3-approximation algorithm that allows the use of bypass disks, where bypass disks are disks that are used as temporary holding points of data. We also looked at the bounded-size matching communication model, where the network does not have unlimited bandwidth. Under this model, we gave an approximation algorithm that takes at most $(2 - 1/B)OPT + \Delta(1 - 1/B)$ rounds for the single-source multicast problem, an approximation algorithm that takes at most $(2 - 1/B)OPT + 3(1 - 1/B)$ rounds for the multi-source broadcast problem, and a $(1 + (3 + o(1))(1 - 1/B))$-approximation algorithm for the multi-source multicast problem. At the end of this chapter, we showed that finding a schedule with minimum number of rounds is NP-hard.

In Chapter 6, we studied problems of broadcasting and multicasting in two-tier communication networks, which arise in Networks of Workstations, grid computing, and clustered wide-area network systems. We first gave a 2-approximation algorithm, called LCF, for this problem. Using this algorithm as a building box, we gave a 2-approximation algorithm for the multicast problem. We also considered two new communication models, bounded degree model and bounded-size matching model, which remove the assumption that the global network has unlimited bandwidth. We gave algorithms with approximation ratio of 3 for both broadcast and multicast problems under the bounded degree model, and algorithms with approximation ratio of 2 for both problems under the bounded-size matching model. We then considered the postal model version of the problems, and showed that LCF Algorithm gives a factor of 3 approximation. We developed a 2-approximation algorithm if the optimal solution are also required to minimize the total number of global transfers. We also presented an experimental study of the effect of having inaccurate information regarding the sizes of the clusters.

There are several open problems regarding the NP-hardness of the problems we considered in this thesis. We only know the multi-source multicast problem of minimizing the makespan is NP-hard. The hardness of the single-source multicast problem is unknown. We believe the multi-source broadcast problem is polynomial time solvable, but no polynomial-time exact algorithm is known. Moreover, we do not know if the broadcasting or multicasting problem in any of the two-tier communication models we considered is NP-hard or not. In Chapters 4 and 5 we studied multi-source broadcasting and multi-source multicasting problems in a local-area network. It would be interesting to develop algorithms to solve these problems under the two-tier communication model. Eventually we want to solve the data migration problem under this model. We would also like to consider other primitive operations like scatter and gather under this model. Moreover, there are several interesting generalizations of the two-tier communication model. For example, we would like to consider the model when the communication time in different clusters may be different due to different speed networks and different speed processors. Another direction of future work is to investigate the effect of variation in the speed of the global network (i.e., the value of C). It is because the total available bandwidth of the global network may change over time. Moreover, the LogP model [24] suggests an alternative framework when dealing with nodes in a single cluster. We would like to have a "two-tier LogP" model with different throughput and latency parameters for the local networks (intra-cluster) and global networks (inter-cluster).

In Chapter 7 we focused on improving the task completion time by re-routing the data through intermediate hosts. We developed a coordinated approach for computing data collection schedules using network flows. We then gave a comprehensive performance study of possible approaches to the data collection problem, and specifically we studied our coordinated method as compared to non-coordinated and direct methods. The performance improvements of our coordinated method are achieved under low storage requirement overheads and without significant detrimental effects on other network traffic throughput. Moreover, we have showed that under mis-estimation situations coordinated methods still perform better than non-coordinated methods. Under high degree of mis-estimation, the performance of coordinated methods are less sensitive to such mis-estimation than the performance of non-coordinated methods. We also showed that coordinated methods has a greater potential for adaptation than non-coordinated methods. Given the graph-theoretic model, our coordinated data transfer algorithm gives an optimal data transfer schedule, with respect to the makespan metric. That is, non-coordinated and direct solutions are part of the feasible solutions within this graph-theoretic model but not necessarily optimal ones. We have also established experimentally (through ns simulations), that the lack of knowledge of the paths provided by the network to send data, are not a significant barrier. Of course, the more we know about the available capacity and paths chosen by the network, the better potentially our modeling can be.

Eventually, we would like to extend the model which also models shared congestion links. Another related question is how to infer the shared link information from end-to-end measurements. Because the transfers may take a long time to finish, and the background traffic between devices may change, we would like to develop some dynamic adaptation schemes to adjust the transfer schedule dynamically. Lastly, we would like to test our coordinated data collection approach under real network environments and incorporate the additional cost of collecting useful information such as the link capacities into our evaluation studies as well as understanding the corresponding effects on our coordinated approach.

BIBLIOGRAPHY

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pages 131–145, Banff, Canada, Oct. 2001.

[3] E. Anderson, J. Hall, J. Hartline, M. Hobbs, A. Karlin, J. Saia, R. Swaminathan, and J. Wilkes. An experimental study of data migration algorithms. In *Proceedings of the 5th International Workshop on Algorithm Engineering*, LNCS 2141, pages 145–158, New York, NY, 2001. Springer-Verlag.

[4] T. Anderson, D. Culler, D. Patterson, and the NOW team. A case for NOW (networks of workstations). *IEEE Micro*, 15(1):54–64, Feb. 1995.

[5] B. R. Badrinath and P. Sudame. Gathercast: An efficient mechanism for multi-point to point aggregation in IP networks. Technical Report DCS-TR-362, Computer Science Department, Rutgers University, Piscataway, NJ, July 1998.

[6] B. Baker and R. Shostak. Gossips and telephones. *Discrete Mathematics*, 2:191–193, 1972.

[7] M. Banikazemi, V. Moorthy, and D. K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *Proceedings of the 1998 International Conference on Parallel Processing*, pages 460–467. IEEE Computer Society, 1998.

[8] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Multicasting in heterogeneous networks. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 448–453, 1998.

[9] A. Bar-Noy and S. Kipnis. Designing broadcast algorithms in the postal model for message-passing systems. *Mathematical Systems Theory*, 27(5), 1994.

[10] J. Bermond, L. Gargano, and S. Perennes. Optimal sequential gossiping by short messages. *Discrete Applied Mathematics*, 86(2–3):145–155, 1998.

[11] J. Bermond, L. Gargano, A. A. Rescigno, and U. Vaccaro. Fast gossiping by short messages. *SIAM Journal on Computing*, 27(4):917–941, 1998.

[12] P. Bhat, C. Raghavendra, and V. Prasanna. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 63(3):251–263, 2003.

[13] S. Bhattacharjee, W. C. Cheng, C.-F. Chou, L. Golubchik, and S. Khuller. Bistro: A platform for building scalable wide-area upload applications. *ACM SIGMETRICS Performance Evaluation Review*, 28(2):29–35, Sep. 2000.

[14] J. A. Bondy and U. S. R. Murty. *Graph Theory with applications.* Elsevier, 1976.

[15] J. Bruck, D. Dolev, C. Ho, M. Rosu, and R. Strong. Efficient message passing interface (MPI) for parallel computing on clusters of workstations. *Journal of Parallel and Distributed Computing*, 40(1):19–34, Jan. 1997.

[16] R. T. Bumby. A problem with telephones. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):13–18, Mar. 1981.

[17] K. Calvert, J. Griffioen, B. Mullins, A. Sehgal, and S. Wen. Concast: Design and implementation of an active network service. *IEEE Trans. on Networking*, 19:426–437, 2000.

[18] R. L. Carter and M. E. Crovella. Measuring bottleneck link speed in packet-switched networks. *Performance Evaluation*, 27–28:297–318, 1996.

[19] W. Cheng, C. Chou, L. Golubchik, S. Khuller, and Y.-C. Wan. A coordinated data collection approach: Design, evaluation, and comparison. *IEEE Journal on Selected Areas in Communications—Special Issue on Design, Implementation and Analysis of Communication Protocols*, 22(10):2004–2018, Dec. 2004.

[20] W. C. Cheng, C.-F. Chou, L. Golubchik, S. Khuller, and Y.C. Wan. On a graph-theoretic approach to scheduling large-scale data transfers. Technical Report CS-TR-4322, University of Maryland, College Park, MD, Jan. 2002.

[21] W.C. Cheng, C.-F. Chou, L. Golubchik, S. Khuller, and Y.-C. Wan. Large-scale data collection: A coordinated approach. In *Proceedings of IEEE INFOCOM*, Mar.–Apr. 2003.

[22] C.-F. Chou, Y.-C. Wan, W. C. Cheng, L. Golubchik, and S. Khuller. A performance study of a large-scale data collection problem. In *Proceedings of the 7th International Workshop on Web Content Caching and Distribution*, pages 259–272, Aug. 2002.

[23] E. J. Cockayne and A. G. Thomason. Optimal multi-message broadcasting in complete graphs. *Utilitas Mathematica*, 18:181–199, 1980.

[24] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12, 1993.

[25] G. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, NJ, 1963.

[26] Y. Dinitz, N. Garg, and M. Goemans. On the single source unsplittable flow problem. In *Proceedings of 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 290–299, 1998.

[27] A. B. Downey. Using pathchar to estimate Internet link characteristics. In *Proceedings of ACM SIGCOMM 1999*, pages 241–250, 1999.

[28] M. Elkin and G. Kortsarz. A sublogarithmic approximation algorithm for the undirected telephone broadcast problem: a path out of a jungle. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 76–85, 2003.

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

[30] A. M. Farley and S. Hedetniemi. Broadcasting in grid graphs. In *Proceedings of the 9th SE Conference on Combinatorics, Graph Theory and Computing*, pages 275–288. Utilitas Mathematica, 1978.

[31] A. M. Farley and A. Proskurowski. Gossiping in grid graphs. *Journal on Combinatorial Information Systems Science*, 5:161–172, 1980.

[32] L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, NJ, 1962.

[33] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 1998.

[34] P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. *Discrete Applied Mathematic*, 53(1-3):79–133, 1994.

[35] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y. Jin. *Internet Distance Map Service.* http://idmaps.eecs.umich.edu/, 1999.

[36] M. R. Garey, R. L. Grahma, and J. D. Ullman. An analysis of some packing algorithms. *Combinatorial Algorithms (Courant Computer Science Symposium, No. 9)*, pages 39–47, 1972.

[37] A. V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms*, 22(1):1–29, 1997.

[38] A. V. Goldberg. *Andrew Goldberg's Network Optimization Library.* http://www.avglab.com/andrew/soft.html, 2001.

[39] L. Golubchik, S. Khanna, S. Khuller, R. Thurimella, and A. Zhu. Approximation algorithms for data placement on parallel disks. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, 2000.

[40] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[41] A. Hajnal, E. C. Milner, and E. Szemeredi. A cure for the telephone disease. *Canadian Mathematical Bulletin*, 15(3):447–450, 1972.

[42] J. Hall, J. Hartline, A. R. Karlin, J. Saia, and J. Wilkes. On algorithms for efficient data migration. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 620–629, 2001.

[43] F. Harary and A. J. Schwenk. The communication problem on graphs and digraphs. *Journal of The Franklin Institute*, 297:491–495, 1974.

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

[45] D.S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing, 1997.

[46] B. Hoppe and É. Tardos. Polynomial time algorithms for some evacuation problems. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 512–521, 1994.

[47] B. Hoppe and É. Tardos. The quickest transshipment problem. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 443–441, 1995.

[48] J. Hromkovic, R. Klasing, B. Monien, and R. Peine. Dissemination of information in interconnection networks (broadcasting and gossiping). In D.-Z. Du and D.F. Hsu, editors, *Combinatorial Network Theory*, pages 125–212. Kluwer Academic Publishers, Netherlands, 1996.

[49] http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html. *Georgia Tech Internetwork topology generator*.

[50] http://www.isi.edu/nsnam/ns/. *The Network Simulator—ns-2.*

[51] C. A. J. Hurkens. Spreading gossip efficiently. *Nieuw Archief voor Wiskunde*, 5(1):208–210, 2000.

[52] P. Husbands and J. Hoe. MPI-StarT: Delivering network performance to numerical applications. In *Proceedings of the IEEE/ACM SC98 Conference*, page 17, Nov. 1998.

[53] V. Jacobson. *pathchar.* http://www.caida.org/tools/utilities/others/pathchar/, 1997.

[54] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[55] R. Karp, A. Sahay, E. Santos, and K. Schauser. Optimal broadcast and summation in the LogP model. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 142–153, 1993.

[56] S. Kashyap and S. Khuller. Algorithms for non-uniform size data placement on parallel disks. In *Proceedings of the 23rd Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, LNCS 2914, pages 265–276, New York, NY, 2003. Springer-Verlag.

[57] S. Khuller and Y. A. Kim. On broadcasting in heterogeneous networks. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1011–1020. Society for Industrial and Applied Mathematics, 2004.

[58] S. Khuller, Y. A. Kim, and Y.-C. Wan. On generalized gossiping and broadcasting. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA'03)*, LNCS 2832, pages 373–384, 2003.

[59] S. Khuller, Y. A. Kim, and Y.-C. Wan. Algorithms for data migration with cloning. *SIAM Journal on Computing*, 33(2):448–461, 2004.

[60] S. Khuller, Y. A. Kim, and Y.-C. Wan. Broadcasting on networks of workstations. Manuscript, 2005.

[61] S. Khuller, Y. A. Kim, and Y.-C. Wan. On generalized gossiping and broadcasting. *Accepted for publications in Journal of Algorithms*, 2005.

[62] S. Khuller, Y. A. Kim, and G. J. Woeginger. Approximation schemes for broadcasting in heterogeneous networks. In *Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'04)*, LNCS 3122, pages 163–170, 2004.

[63] T. Kielmann, H. Bal, and S. Gorlatch. Bandwidth-efficient collective communication for clustered wide area systems. In *Proceedings of the 14th International Symposium on Parallel and Distributed Processing*, page 492, 2000.

[64] T. Kielmann, R. Hofman, H. Bal, A. Plaat, and R. Bhoedjang. MagPIe: MPI's collective communication operations for clustered wide area systems. In *Proceedings of the 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 131–140, 1999.

[65] J. M. Kleinberg. Single-source unsplittable flow. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 68–77, 1996.

[66] W. Knodel. New gossips and telephones. *Discrete Mathematics*, 13:95, 1975.

[67] D. W. Krumme, G. Cybenko, and K. N. Venkataraman. Gossiping in minimal time. *SIAM Journal on Computing*, 21(1):111–139, 1992.

[68] K. Lai and M. Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 123–134, Mar., 2001.

[69] K. Lebensold. Efficient communication by phone calls. *Studies in Applied Mathematics*, 52:345–358, 1973.

[70] H. M. Lee and G. J. Chang. Set to set broadcasting in communication networks. *Discrete Applied Mathematics*, 40(4):411–421, 1992.

[71] T. Leighton. The challenges of delivering content on the Internet. In *Proceedings of ACM SIGMETRICS 2001*, page 214, Cambridge, MA, June 2001.

[72] D. Liben-Nowell. Gossip is synteny: incomplete gossip and an exact algorithm for syntenic distance. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 177–185, 2001.

[73] Pangfeng Liu. Broadcast scheduling optimization for heterogeneous cluster systems. *Journal of Algorithms*, 42(1):135–152, 2002.

[74] B. Lowekamp and A. Beguelin. ECO: Efficient collective operations for communication on heterogeneous networks. In *Proceedings of the 10th International Parallel Processing Symposium*, pages 399–405, Honolulu, HI, Apr. 1996.

[75] B. A. Mah. *pchar*. http://www.kitchenlab.org/www/bmah/Software/pchar/, 2001.

[76] G. De Marco, L. Gargano, and U. Vaccaro. Concurrent multicast in weighted networks. In *Proceedings of the 6th Scandanavian Workshop on Algorithm Theory*, LNCS 1432, pages 193–204. Springer-Verlag, 1998.

[77] Message passing interface forum, Mar. 1994.

[78] K. Moore, J. Cox, and S.Green. SONAR—A network proximity service. *IETF Internet-Draft*, 1996.

[79] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[80] J. Pruyne and M. Livny. Interfacing condor and PVM to harness the cycles of workstation clusters. *Journal on Future Generations of Computer Systems*, 12(1):67–85, 1996.

[81] D. Richards and A. L. Liestman. Generalization of broadcasting and gossiping. *Networks*, 18:125–138, 1988.

[82] D. Rubenstein, J. F. Kurose, and D. F. Towsley. Detecting shared congestion of flows via end-to-end measurement. In *Proceedings of ACM SIGMETRICS 2000*, pages 145–155, 2000.

[83] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: A case for informed Internet routing and transport. *IEEE Micro Magazine*, 19:50–59, Jan. 1999.

[84] S. Savage, A. Collins, and E. Hoffman. The end-to-end effects of Internet path selection. In *Proceedings of ACM SIGCOMM 1999 Conference on Applications, technologies, architectures, and protocols for computer communication*, pages 289–299, 1999.

[85] S. Seshan, M. Stemm, and R.H. Katz. SPAND: Shared passive network performance discovery. In *Proceedings of the 1st USENIX Symp. on Internet Technologies and Systems*, pages 285–294, Dec. 1997.

[86] H. Shachnai and T. Tamir. On two class-constrained versions of the multiple knapsack problem. *Algorithmica*, 29:442–467, 2001.

[87] P. J. Slater, E. Cockayne, and S.T. Hedetniemi. Information dissemination in trees. *SIAM Journal on Computing*, 10:692–701, 1981.

[88] R. Tijdeman. On a telephone problem. *Nieuw Archief voor Wiskunde*, 19(3):188–192, 1971.

[89] V. V. Vazirani. *Approximation Algorithms.* Springer-Verlag, 2001.

[90] V. G. Vizing. On an estimate of the chromatic class of a p-graph (Russian). *Diskret. Analiz.*, 3:25–30, 1964.

[91] R. Wolski and M. Swany. *Network Weather Service.* http://nws.cs.ucsb.edu/, 1999.