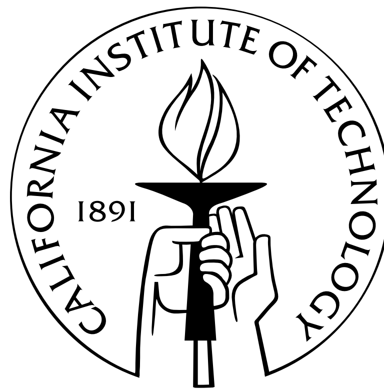
Distributed Averaging and Efficient File Sharing on Peer-to-Peer Networks

Thesis by

Mortada Mehyar

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy



California Institute of Technology
Pasadena, California

2007
(Defended August 21, 2006)

© 2007
Mortada Mehyar
All Rights Reserved

To my mom, dad, and brother.

Contents

Acknowledgements **xiii**

Abstract **xv**

1 Introduction and Background **1**

1.1 What is Peer-to-peer? 1

1.2 A Peer-to-peer Revolution 2

1.2.1 Napster 2

1.2.2 Gnutella 3

1.2.3 DHT-based Systems 5

1.2.4 BitTorrent 5

1.3 What This Thesis Entails 6

2 Distributed Averaging **9**

2.1 Introduction to Distributed Averaging 9

2.1.1 What Is Distributed Averaging? 9

2.1.2 Related Work 10

2.2 Problem Setup 11

2.3 Algorithm A1 13

2.3.1 Convergence of Algorithm A1 15

2.3.2 Implementation and Deadlock Avoidance 21

2.3.3 Simulation of A1 22

2.4 Dynamic Topology: Joining and Leaving of Nodes 23

2.5 Algorithm A2 26

2.5.1 Convergence Rate 32

2.5.2 Experimental Results 32

2.5.3 Simulation of A2 34

2.6 Applications and Extensions 35

2.6.1 Calculating the N-th Moment of Measurement Distributions on Sensor Networks 35

2.6.2 Tracing Dynamic Network Averages 35

2.6.3 Dynamic Node Counting in a Transient Peer-to-peer Network 36

2.6.4 Improving Peer Selection Algorithms in BitTorrent 37

2.7 Summary and Conclusion 38

3 Modelling BitTorrent-like Peer-to-peer File Sharing 39

3.1 Introduction to Peer-to-peer File Sharing 39

3.1.1 Demand for Large Content 39

3.1.2 The BitTorrent Protocol 39

3.2 Model Setup 41

3.3 Last Finish Time 43

3.4 Other Optimality Criteria 46

3.4.1 Average Finish Time 46

3.4.2 Min-Min Finish Times 48

3.5 General Properties 48

3.6 Optimal Average Finish Time 52

3.6.1 Optimal Average when $M = N$ 52

3.6.2 Network of Two Peers 53

3.6.3 Optimal Average when $M = N - 1$ 55

3.6.4 Networks of Three Peers 57

3.6.5 Networks of Four and More Peers 60

3.7 Min-min Finish Times 62

3.8 Selfish Peers 64

3.9 Summary and Conclusion 66

4 Future Directions 69

4.1 Selfish vs. Altruistic Peers 69

4.2 Data Identity vs. Network Coding 70

4.3 Fairness Objectives 70

Bibliography **73**

List of Figures

1.1 An illustration of the schematic of Napster. The central server keeps a directory of what peers have what files. Peer queries the server to obtain the IP addresses of the peers who have files matching the query keywords. 3

1.2 An illustration of the schematic of Gnutella. Unlike Napster, there is no central server that keeps an index of the files. Peers form an overlay network of small connections. A peer can send queries to its neighbors to see if they have any files it desires. If a match is not found locally, the queries will be forwarded and propagate many hops through the network. In this figure, the red arrows represent the original queries. The forwarded queries are represented as the blue arrows. 4

2.1 An example network for distributed averaging. Each node is associated with a value and the goal is to calculate the average of these values in a distributed fashion. 12

2.2 Illustration of the message-passing scheme. 14

2.3 An example network consisting of four nodes in a “star” topology. 17

2.4 The four-node network embedded on the real line according to node value x_i. The bold lines indicate *segments*, i.e., intervals on the real line separating two adjacent values. The dashed curves indicate the communication topology from Figure 2.3. Thus, an update on the link between node 1 and node 3 will *claim* two segments, $[x_3, x_2]$ and $[x_2, x_1]$. 19

2.5 The graph H for the example network, where the node indices are taken as the UIDs. 22

2.6 Three different topologies for simulations of A1. The top left topology is a ring network, the top right is a ring network plus 8 connections, and the top topology is a ring network plus 12 connections. 23

2.7 Simulation results of A1 on a ring topology. On the top graph, each color represents a trajectory of one state value over time. Half of the nodes start with initial value 100 and the other half with 0. Note that all state values eventually converge to the target average of 50. On the bottom, the potential function is also plotted. Note the rapid convergence of the potential function. 24

2.8 Simulation results of A1 on a ring topology plus 8 extra connections. On the top graph, each color represents a trajectory of one state value over time. Again, half of the nodes start with initial value 100 and the other half with 0. The behavior of the potential function over time is plotted on the bottom. . 25

2.9 Simulation results of A1 on a ring topology plus 12 extra connections. On the top graph, each color represents a trajectory of one state value over time. The behavior of the potential function over time is plotted on the bottom. 26

2.10 State histories from a simulation of algorithm A1 on a fifty-node network. Round-trip delays on each link were assigned randomly, between 40 (ms) and 1000 (ms). Note that all states converge towards the average value .5. . . . 27

2.11 An illustration of the joins and leaves of peers on a peer-to-peer network. Existing peers can leave the system either voluntarily or due to failure. New peers can join the network by connecting to any of the existing peers. 28

2.12 A snapshot of the PlanetLab world-wide research network. Our experiments were carried out on overlay networks of these nodes. 33

2.13 Sample histories from an experiment on the PlanetLab network, using 100 nodes and algorithm A2. Round-trip times on this network ranged between tens of milliseconds to approximately one second. Note the rapid convergence of the estimates. 34

2.14 An example network for the counting application of distributed averaging. Notice that one and only one peer has an initial value of 1, and the remaining peers have a value of 0. The average value of this particular network is 1/7, the reciprocal of the number of peers. 37

3.1 A schematic for the BitTorrent protocol. New peers first obtain a torrent file, and then ask the tracker for IP addresses of existing peers to connect to. . . 40

3.2 A schematic for the rarest-first policy employed by the BitTorrent protocol. Pieces that are "rare" in the network have a higher chance of being uploaded than pieces that are commonly found among peers. Here, the top peer chooses to send piece 3 instead of piece 2 to the peer on the bottom right, because piece 3 is less common in the local network than piece 2. 41

3.3 A file segment B is uploaded by three nodes in this graph. It can be regarded as the data traversing three hops in the flow. 49

3.4 A file segment is divided into two disjoint segments of equal size, B_1 and B_2. Notice that each sending node still uses the same upload capacity, and each receiving node still receives the same segments. This flow is now "two-hop." . 49

3.5 An illustration of the finish times that are optimal for average. The parameters chosen are $C_s = 500$, $C_1 = 200$, $C_2 = 80$, and $C_3 = 70$. The optimal values of T_A and T_L are also plotted. 61

3.6 An illustration of the behavior of the finish times that are optimal for average, with different multiplicities. There are three intervals C_s can be in, and they are illustrated on top. The intervals are marked with their corresponding value of multiplicity. An illustration of the different behaviors for different intervals is on the bottom. Notice how the peer finish times "spread out" as C_s increases. The optimal value of the last finish time, T_L^*, is also drawn for comparison. 62

Acknowledgements

I would like to thank my advisor Steven Low for his support all these years. Without his support and encouragement, this thesis simply would not exist. I am grateful for having such a great advisor.

I would like to thank everyone in Netlab, especially Christine Ortega, Raj Jayaram, Hyojeong (Dawn) Choe, Cheng Jin, Ao (Kevin) Tang, Xiaoliang (David) Wei, Lun Li, Jiantao Wang and John Pongsajapan. I would also like to thank the committee members Richard Murray, John Doyle, Mani Chandy and Tracey Ho.

It has been a wonderful experience at Caltech, for being able to work with some of the smartest people I have ever met. I would like to especially thank Demetri Spanos and Weihsin Gu, with whom I have collaborated closely through out the years. Without their inputs, many of the ideas would not have been thought of, and many of the results would not have been discovered.

A lot of what I learned during the years at Caltech is not actually possible to summarize by the equations and theorems in this thesis. I would like to thank all my friends who have made everything more rewarding. I would like to especially thank Sotirios Masmanidis, Jonathan Harel, Jeremy Thorpe, Lin Han, Fuling Yang, Kelvin Yuen for their friendship.

I would like to thank my girl friend Bernice Yeh for all the wonderful memories that we share. Finally, I would like to thank my uncle, my brother, and my parents for their love and continuous support.

Abstract

The work presented in this thesis is mainly divided in two parts. In the first part we study the problem of distributed averaging, which has attracted a lot of interest in the research community in recent years. Our work focuses on the issues of implementing distributed averaging algorithms on peer-to-peer networks such as the Internet. We present algorithms that eliminate the need for global coordination or synchronization, as many other algorithms require, and show mathematical analysis of their convergence.

Discrete-event simulations that verify the theoretical results are presented. We show that the algorithms proposed converge rapidly in practical scenarios. Real-world experiments are also presented to further corroborate these results. We present experiments conducted on the PlanetLab research network. Finally, we present several promising applications of distributed averaging that can be implemented in a wide range of areas of interest.

The second part of this thesis, also related to peer-to-peer networking, is about modelling and understanding peer-to-peer file sharing. The BitTorrent protocol has become one of the most popular peer-to-peer file sharing systems in recent years. Theoretical understanding of the global behavior of BitTorrent and similar peer-to-peer file sharing systems is however not very complete yet. We study a model that requires very simple assumptions yet exhibits a lot structure. We show that it is possible to consider a wide range of performance criteria within the framework, and that the model captures many of the important issues of peer-to-peer file sharing.

We believe the results provide fundamental insights to practical peer-to-peer file sharing systems. We show that many optimization criteria can be studied within our framework. Many new directions of research are also opened up.

Chapter 1

Introduction and Background

1.1 What is Peer-to-peer?

The Internet has become such an integral part of our lives. We find all kinds of information on the Web, and communicate with people through email and instant messaging every day. This thesis that you are reading now is probably more accessible online than in a library.

We have seen revolutions on the Internet ever since it started. The Web has exploded since 1994 and continues to be a significant part of our world. Numerous applications, including voice over IP, streaming video, and electronic commerce are now being carried out on this remarkable network of millions of computers every day.

Peer-to-peer applications have also become an important phenomenon on the Internet in recent years. Since the advent of Napster in 2000, peer-to-peer applications have seen a great deal of demand from users. A large number of different peer-to-peer systems has been developed over the last five years.

Peer-to-peer applications are a set of systems that break away from the traditional client-server paradigm. Conceptually, a peer-to-peer system is a system whose functionality is decentralized among equally capable components. These equally capable components are called "peers" and they can be, for example, the PCs of the users.

It is worth noting that the original Internet was fundamentally designed as a peer-to-peer system in the 1960s [31]. The ARPANET connected the first four nodes (UCLA, UCSB, SRI, and the University of Utah) not in a client-server way, but in a peer-to-peer way. With the commercialization of the Internet, the client-server division became more and more significant, with consumer clients being connected to a set of much more specialized and powerful servers.

The Domain Name Service (DNS) has been an essential part of the Internet for a long time, and it can be considered a peer-to-peer system as well. DNS is the system that maps human-readable domain names into machine-readable IP addresses. Name servers can be both clients and servers, when a query is propagated through the system.

1.2 A Peer-to-peer Revolution

Despite many existing systems that could be regarded as peer-to-peer, Napster is generally considered the first peer-to-peer application which started the revolution in 2000. Many other peer-to-peer file sharing systems appeared after Napster, including Kazaa [20], Gnutella [16], and BitTorrent [10].

Numerous structured peer-to-peer systems have been developed and studied, including Chord [45], Tapestry [51], Pastry [37], and CAN [38]. There has been a great number of peer-to-peer computing systems as well, including SETI@home [40]. It is estimated that peer-to-peer traffic accounts for 60% of the total traffic on the Internet in 2005. We are certainly in an ongoing peer-to-peer revolution.

There have been several generations of peer-to-peer systems. We will give a brief introduction to some of the more important and representative systems in each generation, and discuss their design.

1.2.1 Napster

In very simple terms, Napster is a file sharing system that lets users look up files that reside in other users' PCs, and download them. This is not easily achievable by the design of the Internet, since end users do not have fixed IP addresses. It is therefore not easy to establish direct connections between peers. In the Napster system, a centralized server (or a cluster of centralized servers) maintains a directory of all the files that the connected peers have on their PCs.

In order for a peer to find a file, it sends a query to the Napster server, and the server looks up the directory to figure out if there are peers in the system who have files matching the query string. The server then returns a list of matched files to the peer, and then the peer can decide to download these files by establishing a direct connection to the peers holding these matched files. A schematic of Napster is illustrated in Figure 1.1.

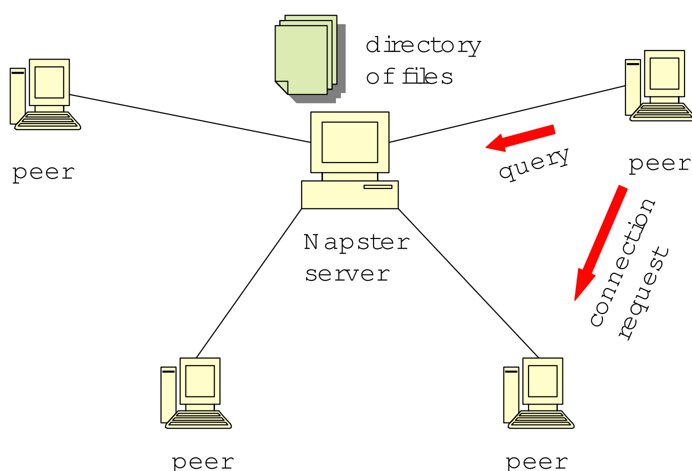


Figure 1.1: An illustration of the schematic of Napster. The central server keeps a directory of what peers have what files. Peer queries the server to obtain the IP addresses of the peers who have files matching the query keywords.

Napster was first released in 1999 and it quickly became popular. It was reported to have millions of users in 2000. The MP3 music downloading on Napster created a lot of copyright issues. Napster was basically shut down in 2001 due to legal pressure from the RIAA, the Recording Industry Association of America [18].

1.2.2 Gnutella

The Gnutella [16] protocol was developed in 2000. Napster's legal demise enhanced the popularity of Gnutella in 2001. Instead of using a centralized directory like Napster, Gnutella replaces directory lookup altogether, and implements the meta-data search by creating an overlay network and flooding queries across the network. LimeWire [25] and BearShare [5] are two of the commercial implementations of the Gnutella protocol.

When a peer wants to search for content, it sends a request to each peer it is connected to. The overlay is designed such that the number of neighbors for each peer is small (typically five). If a peer, upon receipt of a query, can not find any match, it forwards the query to all of *its* neighbors, and so on. This flooding behavior can use up a lot of network resources, and therefore the number of times a query can be forwarded is capped at less than some time-to-live value (typically seven).

If a query matches, the peer who has the match contacts the peer that initiates the query with a response message. The response is usually sent back along the same route the query arrived through. It is therefore possible to cache such search results for later.

If a peer decides to download the matched file, the peer tries to establish a connection directly. If the peer who has the matched file is not behind a firewall, a connection can be established directly and the file can be transferred. However, if the peer with the file is behind a firewall, then that peer will not accept incoming connection requests. In this case the querying peer needs to ask the the server to ask the other peer to initiate the connection instead. Figure 1.2 is a schematic of the Gnutella protocol.

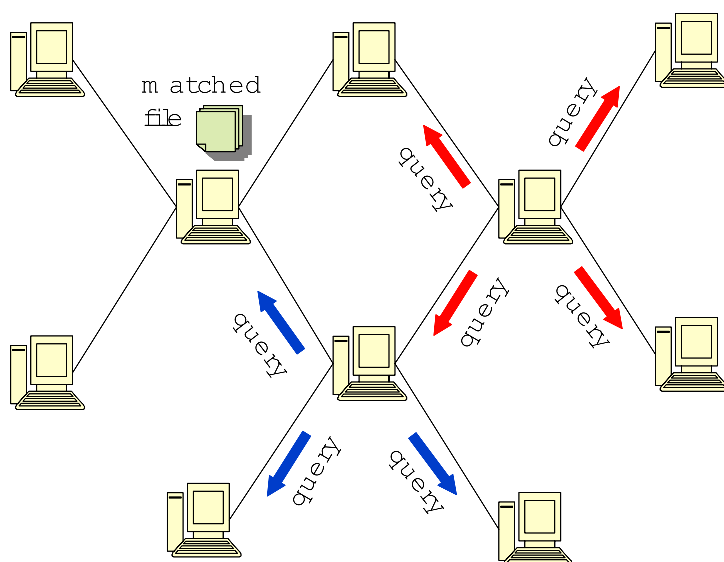


Figure 1.2: An illustration of the schematic of Gnutella. Unlike Napster, there is no central server that keeps an index of the files. Peers form an overlay network of small connections. A peer can send queries to its neighbors to see if they have any files it desires. If a match is not found locally, the queries will be forwarded and propagate many hops through the network. In this figure, the red arrows represent the original queries. The forwarded queries are represented as the blue arrows.

Gnutella is a lot more decentralized than Napster. Napster relies on a centralized directory service, but Gnutella does not have a single point of failure. It is therefore a lot harder to shut down a Gnutella network than a Napster network.

In practice, the flooding search of Gnutella is not very reliable. Flooding can waste a lot of network resources, and it is possible that the queries only propagate to a small

fraction of peers. A system called Gia [8] is proposed to address these issues by replacing Gnutella's flooding algorithm by random walks. Without tight control of overlay topologies, Gnutella is considered to be an *unstructured* system, and it motivated a lot of the structured peer-to-peer systems based on distributed hash tables.

1.2.3 DHT-based Systems

Distributed hash tables (DHTs) implement the functionality of a hash table in a network of peers connected through an application layer overlay [27]. Unlike a centralized hash table, a DHT partitions the ownership of keys to different peers in the system. When values need to be looked up, the DHT system routes queries to peers who are responsible for the keys. DHTs are more scalable than Gnutella's flooding algorithm, but they only support exact-match searching, rather than keyword search of the meta data.

Numerous structured peer-to-peer systems have been developed based on DHTs. Chord [45] is a DHT-based system that hashes each peer and each file into a one-dimensional ring. In an N-node Chord network, each peer maintains $O(log(N))$ number of connections to other peers. Chord guarantees that a query travels no more than a logarithmic number of hops to get to its destination, i.e., the peer who is responsible for answering the query. Chord has been used as a basis for a file system [12] and as a tool to serve DNS [11].

The Content Addressable Network (CAN [38]) uses a fixed constant d and uses a d-dimensional Cartesian coordinate space to implement distributed hashing. The number of connections for each peer to maintain is fixed to be the constant d regardless of the network size, as opposed to $O(log(N))$ in Chord.

Tapestry [51] is another DHT-based system that addresses the issues of locality. Since the application-layer overlays that peer-to-peer systems construct are not necessarily related to the underlying network topology, one hop in the overlay can possibly correspond to many hops in the network layer. Tapestry addresses this issue by taking into account the *locality* of each peer, when forming the routing overlay. Tapestry has enabled the deployment of storage applications such as OceanStore [24].

1.2.4 BitTorrent

In addition to the unscalable flooding search, Gnutella suffers from the problem of free-riding [1]. Free riding occurs when a peer shares the resources of a peer-to-peer network, but does

not contribute to other peers altruistically. It is reported that 70% of Gnutella peers share no files, and nearly 50% of all responses are returned by the top 1% of sharing peers. Free riding is therefore a very serious problem that can lead to performance degradation of the entire Gnutella system.

The essential cause of the free riding problem on Gnutella is that there is no incentive mechanism to encourage sharing. A Gnutella peer does not benefit itself by uploading to other peers in the network. The BitTorrent protocol aims to eliminate the free-riding problem by creating an incentive for peers to share [10, 17]. BitTorrent was introduced in 2001 and became extremely popular by 2003.

Unlike Napster and Gnutella, BitTorrent separates the meta-data search from its file-sharing protocol completely. There is no notion of searching for a file in a BitTorrent network, since there is a separate BitTorrent overlay network (also called a "swarm") for each file. In BitTorrent, the problem of searching for a file therefore becomes a problem of searching for a *network* that distributes such a file. A peer can join a BitTorrent network for the desired file after it obtains a "torrent" file, which is something that contains information about where to find the network, and it is usually published on the Web.

BitTorrent creates an incentive for peers to share by its "tit-for-tat" algorithm. The basic idea is for each peer to reciprocate to the peers who have uploaded to it most. Therefore the more a peer uploads, the more chances other peers would choose to upload to it. BitTorrent therefore creates an incentive for altruistic sharing. In Chapter 3, we will discuss in more detail how BitTorrent works, and we will study a file-sharing model that is closely related to BitTorrent.

1.3 What This Thesis Entails

Decentralized designs are desirable in many situations for their simplicity, robustness, and fault-tolerance. However, simple interactions between the components can often result in complex global behavior. It is therefore very important to understand the behavior of the system as a whole.

TCP congestion control, for example, is a distributed way of regulating sending rates of TCP connections. When TCP connections share network resources and interact, complex global behavior can arise. It turns out that the whole system can be understood as a

global optimization problem [21, 26]. Optimization techniques can be extremely useful for understanding distributed systems. We make use of many methods and ideas from optimization theory in this work.

In this thesis, we study problems related to peer-to-peer computation and peer-to-peer file sharing. In Chapter 2 we present our work on distributed averaging and show that it can be applied to many applications on different kinds of networks. Distributed averaging is a versatile method for many kinds of peer-to-peer computations. We believe distributed averaging can be a promising framework for numerous applications.

In Chapter 3, we study a peer-to-peer file sharing model and show that it can provide new insights to practical systems such as BitTorrent. We show that despite the simplicity of the model, it captures many of the important issues and exhibits a rich structure. We show that it is possible to study different kinds of design objectives for file sharing under this framework. The peer-to-peer file sharing model that we study opens up a new set of research problems, and we believe it provides fundamental understanding of file-sharing systems.

In Chapter 4, we discuss various future directions of the peer-to-peer file-sharing model.

Chapter 2

Distributed Averaging

2.1 Introduction to Distributed Averaging

2.1.1 What Is Distributed Averaging?

“Distributed averaging” is a distributed iterative procedure for calculating averages over a network. This style of asynchronous computing has seen a renewed interest in recent years. While asynchronous iterative computing is not new in itself (see the classic references of Bertsekas and Tsitsiklis [6] and Lynch [28]), new issues arise when one attempts to implement such schemes on unstructured, packet-switched, communication networks such as the Internet.

Averaging serves as a useful prototype for asynchronous iterative computations both because of its simplicity, and its applicability to a wide range of problems. On a sensor network, one may be interested in calculating the average of physical measurements over the entire network. In problems that concern vehicle formation, the quantities being averaged can be the coordinates of the vehicles, and the average can represent the center of mass of the whole system. A network of Web servers may wish to calculate the average processor load, in order to implement some load-balancing scheme. This is currently usually done with dedicated centralized load-balancers, but it is conceivable a distributed solution can be beneficial in many aspects.

A peer-to-peer file-sharing system on the Internet (e.g., BitTorrent [10] and Kazaa [20]) may wish to compute other application-specific averages as well. In peer-to-peer systems, it is usually desirable to eliminate centralized components as much as possible. However, it is also important for a peer-to-peer system to know some global information about the

network. We believe distributed averaging can be a very powerful tool that enables peer-to-peer systems to extract global information in a decentralized fashion.

In principle, one can choose to calculate averages by flooding the entire network with all the values, in order to calculate the average. It is also possible to use a structured messaging scheme over a specially designed overlay topology (e.g., a spanning tree). These are both natural methods for calculating averages on a network, but the former has very large messaging complexity, and the latter has very little flexibility for dynamic topology changes. More importantly, these possible alternative solutions usually require global exchange of information. While it is not clear that this is necessarily a problem in the applications we have discussed above, it seems likely that a distributed scheme involving only local exchange may be desirable.

The iterative procedure of distributed averaging usually does not provide an exact average, but asymptotic convergence to the average instead. In many scenarios an exact average is not required, and one may be willing to trade precision for simplicity. The scalability, robustness, and fault-tolerance associated with distributed averaging can be superior in many situations where exact averaging is not essential. These schemes also resolve issues of global information exchange, as they only require communication among local neighbors.

In this paper, we will present two distributed averaging algorithms and show their convergence in a general asynchronous environment. Our analysis is verified by simulation and experiments on a real-world TCP/IP network. In combination, these results show that the method proposed is both analytically understandable and practically implementable.

2.1.2 Related Work

Much recent research has focused on various distributed iterative algorithms. Distributed averaging, also known as the distributed consensus problem, has been studied in the context of vehicle formation control by Fax and Murray (e.g., [14, 34]). Similar algorithms have been applied to sensor fusion by Spanos, Olfati-Saber, and Murray ([42], [44]), as well as Xiao, Boyd, and Lall [49]). Previous works in the gossip algorithm context utilize probabilistic frameworks and analyze global behavior (see the work of Boyd et al. [7], Kempe et al. [23, 22], and references therein). The "agreement algorithm" was proposed in the work of Tsitsiklis [46] and [47], and it is concerned with letting a distributed set of processors converge to some common value.

Other authors have considered similar iterative mechanisms, including Xiao and Boyd [48], who examine the possibility of link weight optimization for maximizing the convergence rate of the algorithm. The issues of asynchronism have also been studied. An asynchronous time model is analyzed in Boyd et al. [7] which assumes that each node has a clock ticking at the times of a Poisson process. In comparison, the asynchronous model we will use in this paper does not assume any stochastic properties. Also, we do *not* assume that neighboring nodes can simultaneously exchange values, as is implicit in Boyd et al [7].

The Push-Sum algorithm (discussed in Kempe et al. [22]) is a gossip algorithm that can be used to calculate sums and averages on a network. In the synchronous settings of Push-Sum, some probabilistic characterization of the convergence rate can be obtained. The convergence rate of Push-Sum is shown in [22] to depend on the logarithm of the size of the network. In comparison to Push-Sum, our algorithms do not assume a gossip-like randomized communication scheme. Instead, we propose message-passing mechanisms to enable communication among nodes. Also, the convergence rate of our algorithms does not in general depend on the size of the network, but only on the algebraic connectivity of the network. These points will be explained in detail in the later sections.

2.2 Problem Setup

Consider a network, modeled as a *connected* undirected graph $G = (V, E)$. We refer to the vertices (elements of V) as nodes, and the edges (elements of E) as links. The nodes are labeled $i = 1, 2, \ldots, n$, and a link between nodes i and j is denoted by ij.

Each node has some associated numerical value, say z_i, that we wish to average over the network. We will refer to the vector $\mathbf{z}$ whose ith component is z_i. Each node on the network also maintains a dynamic variable x_i, initially set to the static value z_i. We call x_i the *state* of the node i.

When we wish to show the time dependence, we will use the notation $x_i(t)$. We use the notation $\mathbf{x}$ to denote the vector whose components are the x_i terms. Intuitively each node's state $x_i(t)$ is its current estimate of the average value $\sum_{i=1}^{n} z_i/n$. The goal of the averaging algorithm is to let *all* states $x_i(t)$ go to the average $\sum_{i=1}^{n} z_i/n$, as $t \to \infty$.

An example of a network of nodes and their values is shown in Figure 2.1. Notice we assume that the network is a connected graph, and the initial value can be any value for

each node.

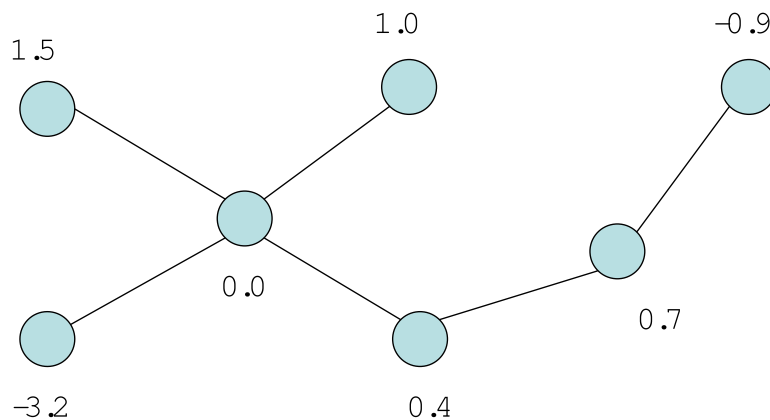


Figure 2.1: An example network for distributed averaging. Each node is associated with a value and the goal is to calculate the average of these values in a distributed fashion.

The work of Olfati-Saber and Murray [34] proposes the following discrete-time system as a mechanism for calculating averages in a network:

$$\mathbf{x}(t+1) = \mathbf{x}(t) - \gamma L\mathbf{x}(t), \tag{2.1}$$

where γ is a stepsize parameter, and L is the *Laplacian* matrix associated with the undirected graph G (see, e.g., [30].)

The Laplacian matrix L is defined as

$$L_{ij} = \begin{cases} d_i & \text{if } i = j, \\ -1 & \text{if there is a link between } i \text{ and } j, \\ 0 & \text{otherwise,} \end{cases}$$

where d_i is the degree or the number of neighbors node i has. The algorithm (2.1) can be viewed as an iterative gradient method for solving the following optimization problem:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbf{R}^n} \quad & \tfrac{1}{2}\mathbf{x}^T L\mathbf{x} \\ \text{s.t.} \quad & \textstyle\sum_i x_i = \sum_i z_i. \end{aligned}$$

Therefore it is not hard to show that this algorithm drives all states x_i to the average, provided the stepsize γ satisfies

$$0 < \gamma < \frac{1}{2d_{\max}},$$

where $d_{\max}$ is the maximum of all the node degrees d_i.

These results implicitly assume synchronization, however. Real-world networks such as the Internet constitute an inherently asynchronous environment with dynamic network delays. Synchronization is usually very hard to achieve, and it is in most cases impractical and undesirable. Another problem with the algorithm (2.1) is that each node must use exactly the same stepsize. The allowable stepsize bound depends on global properties of the network. This information is not available locally and therefore global coordination must be involved.

Without synchronization, nodes cannot update their values exactly at the same time in order to carry out algorithms such as (2.1). Instead, state information from other nodes can only be obtained through the exchange of messages. Therefore, we will need to have a message-passing scheme if we hope to calculate the average on a real network.

In the following sections, we will present our algorithms along with their message-passing schemes and convergence analysis.

2.3 Algorithm A1

In this section we will introduce our first algorithm, A1. At each node i there is a local stepsize parameter γ_i, $0 < \gamma_i < 1$, upon which the node's computation is based. These parameters do not need to be coordinated globally.

The basic "unit" of communication in our scheme is a pairwise update between two nodes. We require two (distinguishable) types of messages, identified in a header. We refer to these two types as *state* messages and *reply* messages. An update is initiated whenever a node sends out a state message containing the current value of its state.

An overview of the message-passing scheme that will enable the pairwise update is as follows:

MP1: At some time, node i initiates a *state* message containing its current state value to some other node j. At some later time, node j receives this message.

MP2: Node j implements a local computation based on the value it receives. It records the result of this computation in a *reply* message and sends this message back to node i.

MP3: At some later time, node i receives j's reply and implements a local computation based on the content of the reply message.

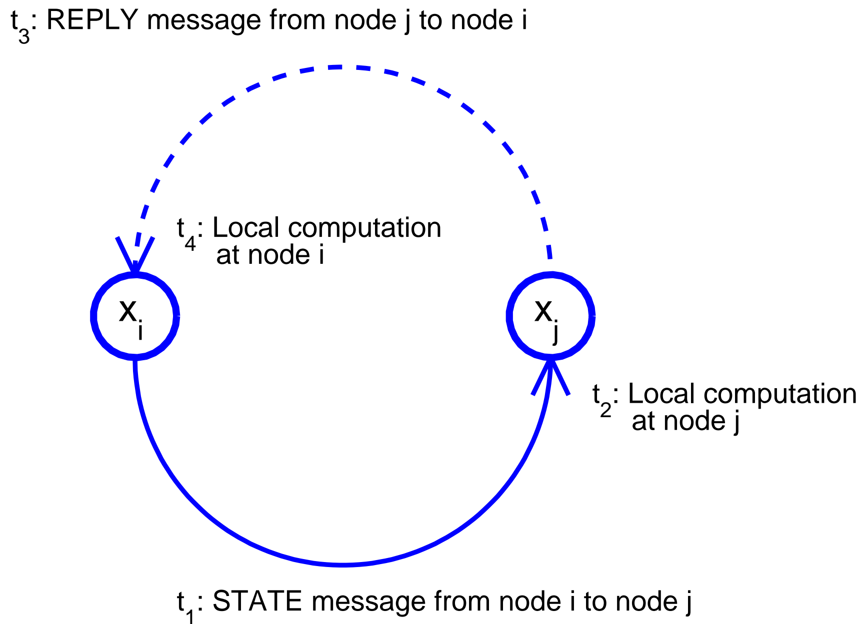


Figure 2.2: Illustration of the message-passing scheme.

In addition to the message-passing scheme, in order to make sure that communications between different pairs of nodes will not interfere, we require that the nodes implement *blocking*. Whenever a node sends out a state message, it blocks any other incoming state messages until it receives a reply from the receiver. If a node receives other STATE messages while it is blocking, it sends back a negative acknowledgement (NACK) indicating that it already has a pairwise update in progress. It also does not initiate any other updates while blocking.

Whenever a node receives a NACK, the update terminates prematurely with no effect on either of the local variables, and the node stops blocking. With the blocking mechanism

in place, a pairwise update is specified as follows:

PW1: Node j receives a state message from node i. If it is blocking, it does nothing and sends a NACK to node i.

PW2: Otherwise, it sends a reply message containing the numerical value $\gamma_j(x_i - x_j)$ to node i and then implements $x_j \leftarrow x_j + \gamma_j(x_i - x_j)$.

PW3: Node i receives the reply message and implements $x_i \leftarrow x_i - \gamma_j(x_i - x_j)$.

Note that node i does not need to know γ_j; all it needs to know is how much change node j has made, which is contained in the reply message. Also note that at the end of a pairwise update, node i has exactly compensated the action of node j, in the sense that the *sum of the states is conserved.*

For the moment, we do not specify the timing or triggering for this event; we will propose one possible scheme (implementation) in section 2.3.2. We will merely make the following assumption:

Eventual Update Assumption: for any link ij and any time t, there exists a later time $t_l > t$ such that there is an update on link ij at time t_l.

This assumption is very similar to the totally asynchronous timing model in [6]. It turns out that this very general asynchronous timing assumption is sufficient to guarantee convergence of the state values to the correct average under algorithm A1.

2.3.1 Convergence of Algorithm A1

Because of the blocking behavior, updates that happen on one link will never interfere with updates on another. This generates a property that is very useful for analysis:

> *With blocking, although updates on different links can span overlapping time intervals, the resulting state values of the network at the conclusion of each pairwise update will be as if the updates were non-overlapping, and therefore sequential in time.*

Thus, aside from the timing details of when updates are initiated, it is equivalent to consider a sequence of pairwise updates enumerated in discrete time $T = \{0, 1, 2, ...\}$, and there is only one update at each time instant. We will do so in the analysis to follow. The evolution of each state x_i under A1 can therefore be understood by considering the following update equations:

$$\begin{cases} x_i(t+1) = (1-\gamma_j)x_i(t) + \gamma_j x_j(t), \\ x_j(t+1) = \gamma_j x_i(t) + (1-\gamma_j)x_j(t), \\ x_k(t+1) = x_k(t),\ \forall k \neq i, j, \end{cases} \tag{2.2}$$

where x_j is the receiver and thus its local γ_j is used instead of γ_i.

Theorem 1 *If the Eventual Update Assumption is satisfied, the A1 algorithm guarantees that*

$$\lim_{t\to\infty} x_i(t) = \frac{1}{n}\sum_{i=1}^{n} z_i, \forall i \in \{1, 2, ..., n\}, \tag{2.3}$$

i.e., all node states converge to the average of the initial states of the network.

Similar convergence results can be found in [47] in the context of agreement algorithms. Our proof of Theorem 1 will make use of the following "potential" function:

$$P(t) = \sum_{\forall (i,j)} |x_i(t) - x_j(t)|, \tag{2.4}$$

where the sum is over all $\frac{n(n-1)}{2}$ possible pairs (i, j). For instance, the potential function for the network in Figure 2.3 is

$$\begin{aligned} &|x_1 - x_2| + |x_1 - x_3| + |x_1 - x_4| \\ &+ |x_2 - x_3| + |x_2 - x_4| + |x_3 - x_4|. \end{aligned}$$

It can be shown that the potential function decreases in the following manner.

Lemma 1 *If nodes (i, j) update at time t with node i being the sender, then at the next*

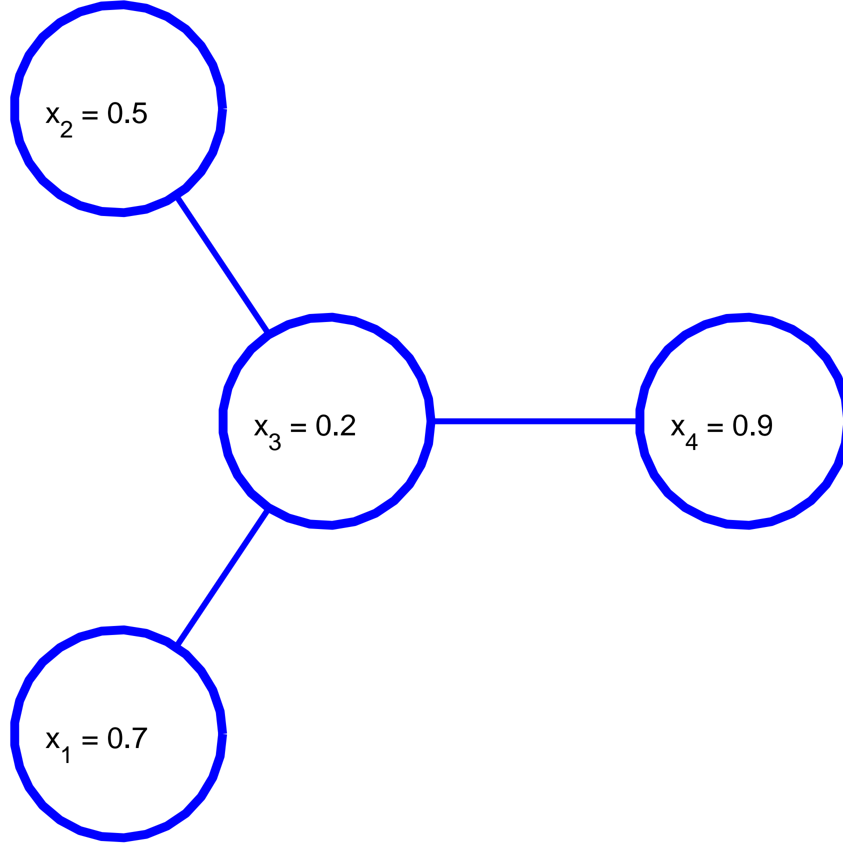


Figure 2.3: An example network consisting of four nodes in a "star" topology.

time unit $t + 1$

$$P(t+1) \leq P(t) - 2\min\{\gamma_j, 1-\gamma_j\}|x_i(t) - x_j(t)|. \tag{2.5}$$

Proof 1 *We can see from (2.2) that besides the term* $|x_i - x_j|$*,* $n-2$ *terms of the form* $|x_k - x_j|$ *and* $n-2$ *terms of the form* $|x_i - x_k|$*,* $k \neq i, j$ *in the potential function* $P(t)$*, are affected by the update. We also have*

$$|x_i(t+1) - x_j(t+1)| = |(1-2\gamma_j)||x_i(t) - x_j(t)|. \tag{2.6}$$

Now consider the sum of two of the affected terms $|x_k(t) - x_i(t)| + |x_k(t) - x_j(t)|$*. If we look at the relative positions of* $x_i(t)$*,* $x_j(t)$*, and* $x_k(t)$ *on the real line, then* either x_k *is in between* x_i *and* x_j or *it is not. Therefore as long as* $0 < \gamma_i < 1$*, it is clear geometrically in*

both cases that we have

$$|x_k(t+1) - x_i(t+1)| + |x_k(t+1) - x_j(t+1)| \\ \leq |x_k(t) - x_i(t)| + |x_k(t) - x_j(t)|.$$

Therefore, together with (2.4) and (2.6) we have

$$\begin{aligned} P(t+1) - P(t) &\leq |x_i(t+1) - x_j(t+1)| \\ &\quad -|x_i(t) - x_j(t)| \\ &\leq -2\min\{\gamma_j, 1-\gamma_j\}|x_i(t) - x_j(t)|. \end{aligned}$$

The quantity $\min\{\gamma_j, 1-\gamma_j\}$ can be thought of as an effective stepsize for node j since a stepsize of .6, say, is equivalent to .4 in terms of reducing the relative difference in absolute value.

Lemma 2 *At any time t, there exists a later time $t' > t$ such that at time t' there has been at least one update on every link since time t. Furthermore,*

$$P(t') \leq \left(1 - \frac{8\gamma^*}{n^2}\right) P(t), \tag{2.7}$$

where $\gamma^* = \min_i \min\{\gamma_i, 1-\gamma_i\}$.

Proof 2 *Without loss of generality, suppose at time t we have $x_1(t) \leq x_2(t) \leq ... \leq x_n(t)$. We call the $n-1$ terms of the form $|x_i(t) - x_{i+1}(t)|$, $i \in \{1, 2, ..., n-1\}$, segments of the network at time t. By expanding every term in the potential function as a sum of segments, we see that the potential function can be written as a linear combination of all the segments:*

$$P(t) = \sum_{i=1}^{n-1} (n-i)i\,|x_i(t) - x_{i+1}(t)|\,. \tag{2.8}$$

We say that a segment $|x_i(t) - x_{i+1}(t)|$ at time t is claimed *at time $t' > t$, if there is an update on a link of nodes r and s such that the interval $[x_s(t'), x_r(t')]$ (on the real line) contains the interval $[x_i(t), x_j(t)]$. For instance, for the network in Figure 2.3, the segments are $|x_3 - x_2|$, $|x_2 - x_1|$, and $|x_1 - x_4|$, as shown in Figure 2.4. Thus, an update on the link*

between node 1 and node 3 will claim segments $[x_3, x_2]$ *and* $[x_2, x_1]$.

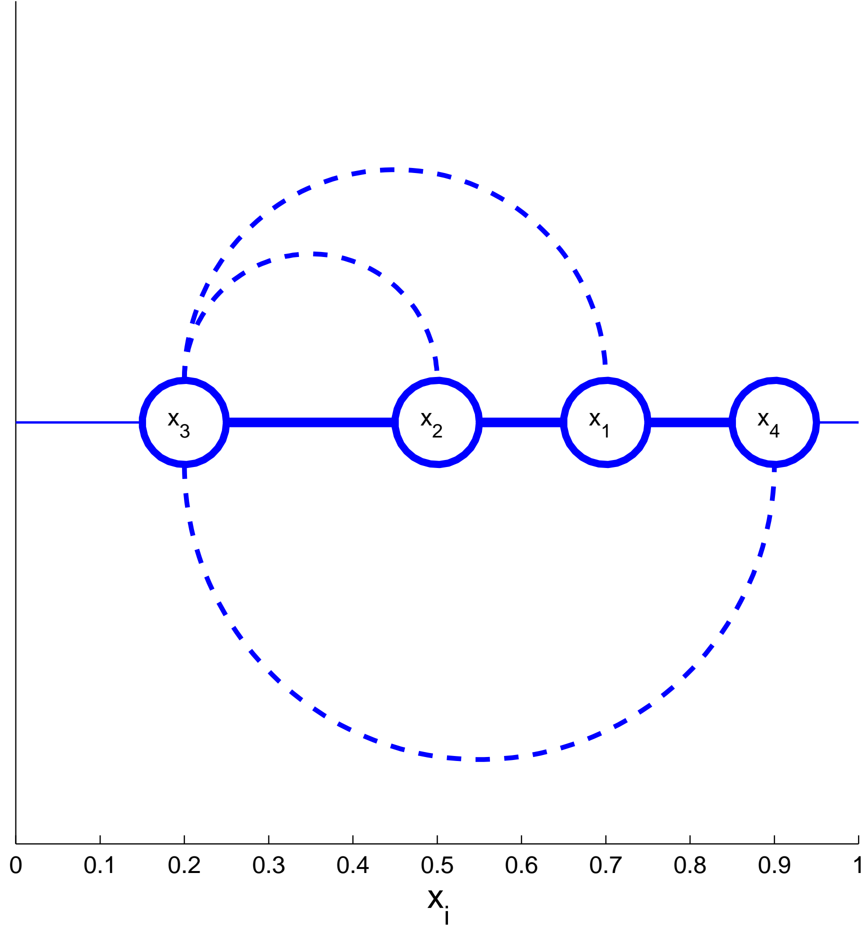


Figure 2.4: The four-node network embedded on the real line according to node value x_i. The bold lines indicate *segments*, i.e., intervals on the real line separating two adjacent values. The dashed curves indicate the communication topology from Figure 2.3. Thus, an update on the link between node 1 and node 3 will *claim* two segments, $[x_3, x_2]$ and $[x_2, x_1]$.

Clearly by using the Eventual Update Assumption on each link, the existence of t' *is guaranteed. From Lemma 1 it is clear that whenever a segment is claimed, it contributes a reduction in the potential function proportional to its size (see (2.5)). Referring to Figure 2.4, it is clear an update that does not claim a segment can only leave the segment unchanged or make it larger. Therefore, no matter* when *a segment is claimed after time* t*, it will contribute at least* $2\gamma^*|x_i(t) - x_{i+1}(t)|$ *reduction in the potential function.*

Now connectedness of the network implies that for each segment there is at least one link such that an update on that link will claim the segment. Therefore, by time t' all *segments will be claimed. Thus the total reduction in the potential function between* t *and* t' *is at least*

$$2\gamma^* \sum_{i=1}^{n-1} |x_i(t) - x_{i+1}(t)|.$$

It follows that

$$\begin{aligned} P(t') &\le P(t) - 2\gamma^* \sum_{i=1}^{n-1} |x_i(t) - x_{i+1}(t)| \\ &= \left(1 - \frac{\sum_{i=1}^{n-1} 2\gamma^* |x_i(t) - x_{i+1}(t)|}{\sum_{i=1}^{n-1} (n-i)i |x_i(t) - x_{i+1}(t)|}\right) P(t) \\ &\le \left(1 - \frac{8\gamma^*}{n^2}\right) P(t), \end{aligned}$$

where in the last inequality we use the fact that $i(n-i) \le n^2/4$.

Proof 3 *(of Theorem 1) Repeatedly applying Lemma 2, we see that*

$$\lim_{t\to\infty} P(t) = 0. \tag{2.9}$$

Therefore

$$\lim_{t\to\infty} |x_i(t) - x_j(t)| = 0, \forall i, j. \tag{2.10}$$

Now by the conservation property (which can be derived from (2.2))

$$\sum_{i=1}^{n} x_i(t) = \sum_{i=1}^{n} z_i, \forall t, \tag{2.11}$$

we see that

$$\lim_{t\to\infty} x_i(t) = \frac{1}{n} \sum_{i=1}^{n} z_i. \tag{2.12}$$

2.3.2 Implementation and Deadlock Avoidance

Any implementation that satisfies the Eventual Update Assumption is within the scope of the convergence proof of A1. However, we have not, as yet, indicated a specific mechanism for the update triggering. Caution must be taken because of the blocking behavior. Without a properly designed procedure for initiating communication, the system can drive itself into a deadlock.

Below we present one particular implementation based on a round-robin initiation pattern, which provably prevents deadlock and satisfies the updating assumption. This is by no means the only way to carry this out, but it has the advantage of being simple and easy to implement.

Our implementation will be based on a unique identifier (UID) for each node in the network. The UIDs must be orderable between different nodes. The UIDs can be obtained, for example, by mapping the IP addresses of the nodes to unique natural numbers. Based on these UIDs, we impose an additional directed graph $H = (V, F)$, in which an edge points from i to j if and only if node j has a higher UID than node i.

This graph has two important properties:

H1: H has at least one root, i.e., a node with no inbound edges.

H2: H is acyclic.

An example is illustrated for our four-node network in Figure 2.5. This graph essentially defines a *sender-receiver relation on each link*.

Our proposed initiation scheme is as follows:

RR1: A node will wait to receive a STATE message from *all* of its inbound edges.

RR2: After having received at least one message from all its inbound edges, the node will then sequentially send a STATE message to each of its outbound edges, ordered by UID.

RR3: Upon completion, it repeats, waiting for all of its inbound edges and so on.

Lemma 3 *The above procedure guarantees that the Eventual Update Assumption is satisfied.*

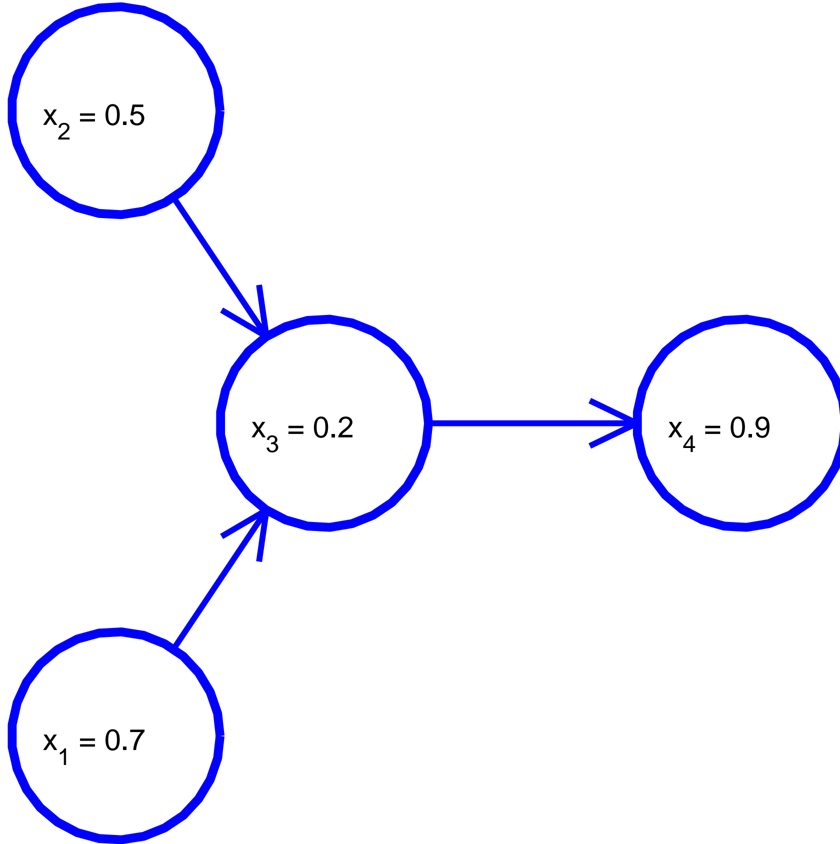


Figure 2.5: The graph H for the example network, where the node indices are taken as the UIDs.

We will prove this by contradiction. Suppose there is a link ij, with i being the sender, and an interval $[t, \infty)$ during which this link does not carry any message. Then, node i must be waiting for one of its inbound edges to send, implying the existence of a node k with a UID lower than that of i, which is also waiting for one of its inbound edges to send. Repeating this argument, and utilizing the fact that H is acyclic, we can find a path of inactive edges beginning at a root. However, a root has no inbound edges, and hence *must* send to all of its outbound edges at some point in $[t, \infty)$. This is a contradiction, and proves the desired result.

2.3.3 Simulation of A1

We have written a discrete event simulator in Java and simulated algorithm A1. We first show a set of simple simulations based on three different types of topologies as shown in Figure 2.6. The topologies are a ring network, a ring network plus 8 more connections,

and a ring network plus 12 more connections. We observe numerically that increasing the connectivity of the graph generally increases the the convergence speed.

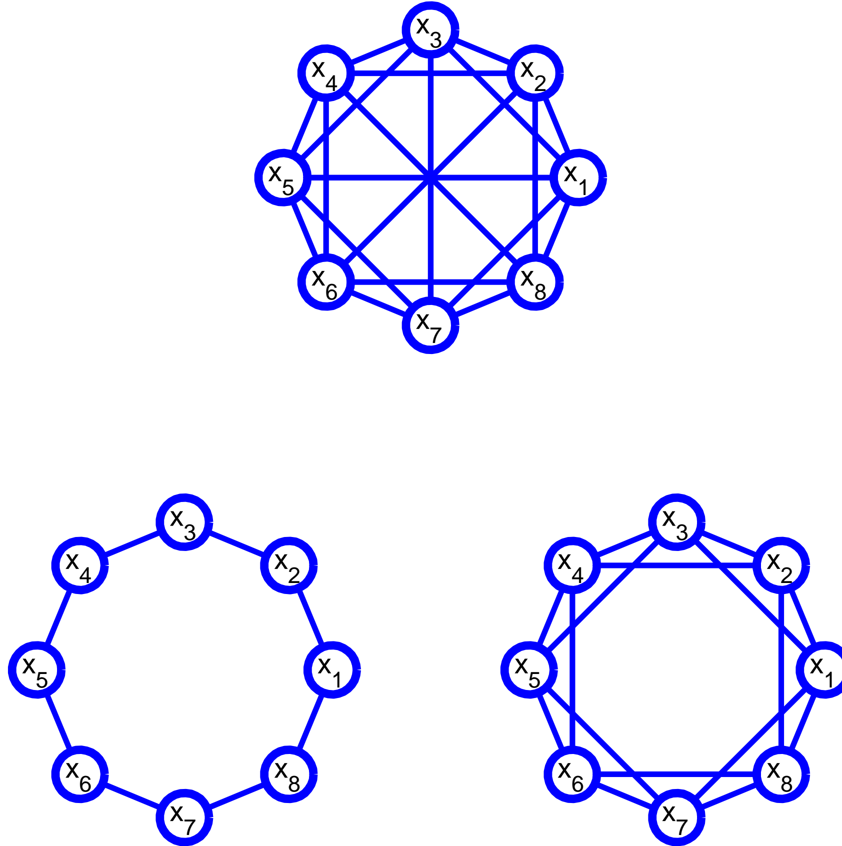


Figure 2.6: Three different topologies for simulations of A1. The top left topology is a ring network, the top right is a ring network plus 8 connections, and the top topology is a ring network plus 12 connections.

We present a simulation of A1 with 50 nodes on a random topology with maximum degree 5. The stepsizes were chosen to be .5 for all nodes and the round-trip delays on the links were uniformly randomly distributed from 40(ms) to 1000(ms). Half of the nodes started with initial states 0 and the others with 1; the target average was therefore .5. The results of this simulation are shown in Figure 2.10.

2.4 Dynamic Topology: Joining and Leaving of Nodes

The A1 algorithm we have described is general enough to accommodate various extensions. In this section we discuss how to handle dynamic network membership (dynamic network

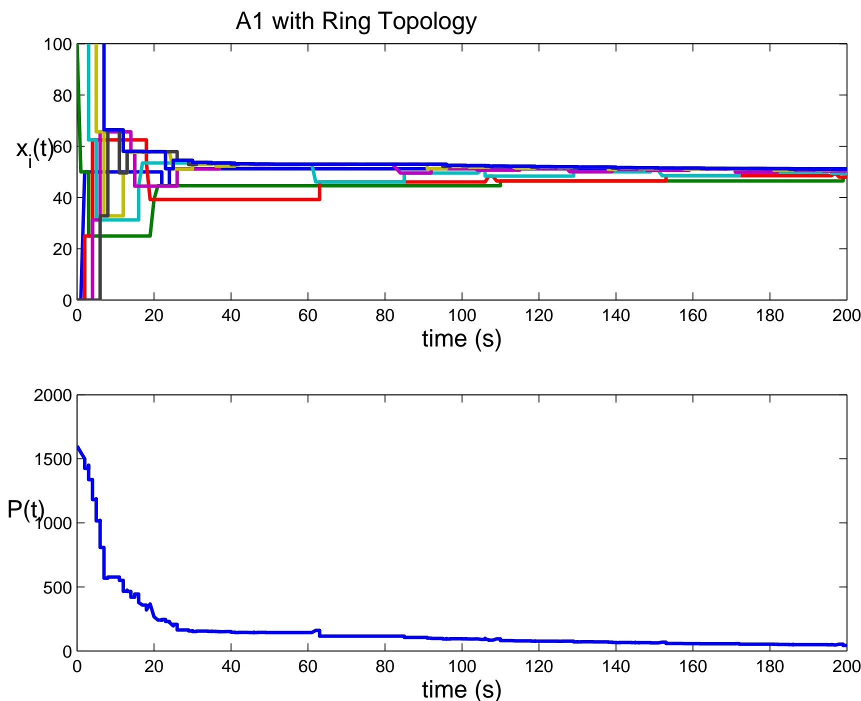


Figure 2.7: Simulation results of A1 on a ring topology. On the top graph, each color represents a trajectory of one state value over time. Half of the nodes start with initial value 100 and the other half with 0. Note that all state values eventually converge to the target average of 50. On the bottom, the potential function is also plotted. Note the rapid convergence of the potential function.

topology), where new nodes can join the averaging network, and current nodes can decide to leave or fail gracefully.

On a peer-to-peer network, it is common for new nodes to join and existing nodes to leave (or fail out of) the system. Due to the extremely transient nature of peer-to-peer networks, one may wish to apply the averaging scheme that allows nodes to join and leave at various points in time. Figure 2.11 illustrates such "joins and leaves" behavior of the network.

A simple mechanism for doing so is for each node to maintain an additional variable associated with each neighbor, denoted by δ_{ij}, which accounts for all the changes made on behalf of that neighbor. This idea is described in [43].

Specifically, each time node i and j interact, the net change in i's state is added to the variable δ_{ij}. Then, if a node leaves the network, all its neighbors subtract δ_{ij} from their current states. It can be shown that this ensures the following conservation property: at any given time, the sum of the states x_i over any connected component of the network

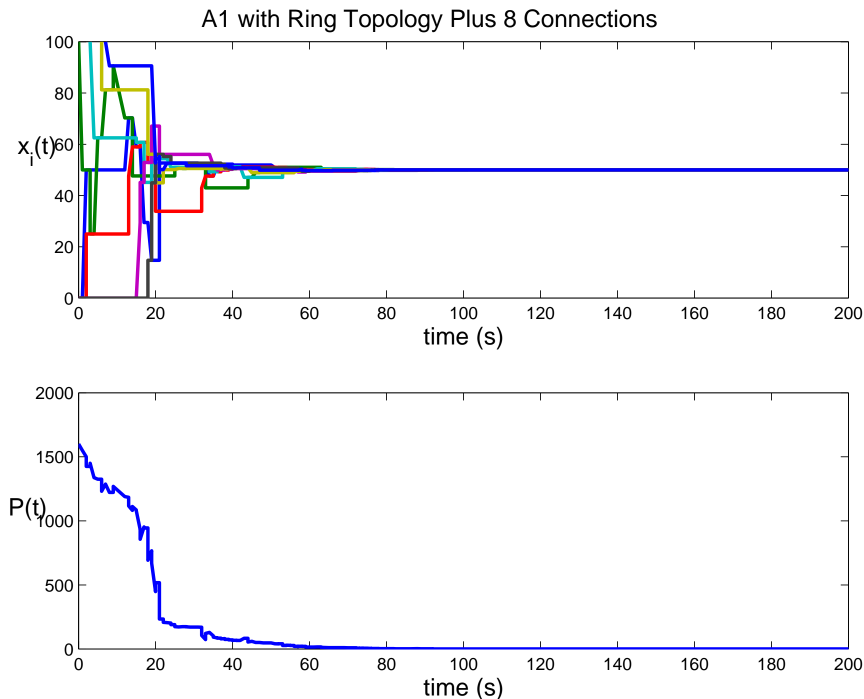


Figure 2.8: Simulation results of A1 on a ring topology plus 8 extra connections. On the top graph, each color represents a trajectory of one state value over time. Again, half of the nodes start with initial value 100 and the other half with 0. The behavior of the potential function over time is plotted on the bottom.

is precisely equal to the sum of the initial values z_i. Thus, after a topology change, the iterative algorithm again begins converging toward the appropriate average quantity over the new network. Note that this serves as a reactive mechanism for node failures, since the failing node's neighbors can detect its failure and compensate by subtracting the associated δ_{ij} terms.

One promising application of the averaging algorithm with dynamic membership is counting the number of nodes on a peer-to-peer network. It is known that due to the transient nature of peer-to-peer nodes, it is often hard to obtain a good estimate of the total number of active nodes on the network. Suppose it can be ensured that one and only one node has set $z_i = 1$, while all others have set $z_i = 0$. If *all* nodes are completely identical, this coordination would be hard to achieve. This is possible, however, on peer-to-peer networks with a bootstrapping server, since the bootstrapping server can be the special node with $z_i = 1$. Then the averaging algorithm, combined with the aforementioned dynamic membership handling, can track the average state of all active nodes $1/n$, the

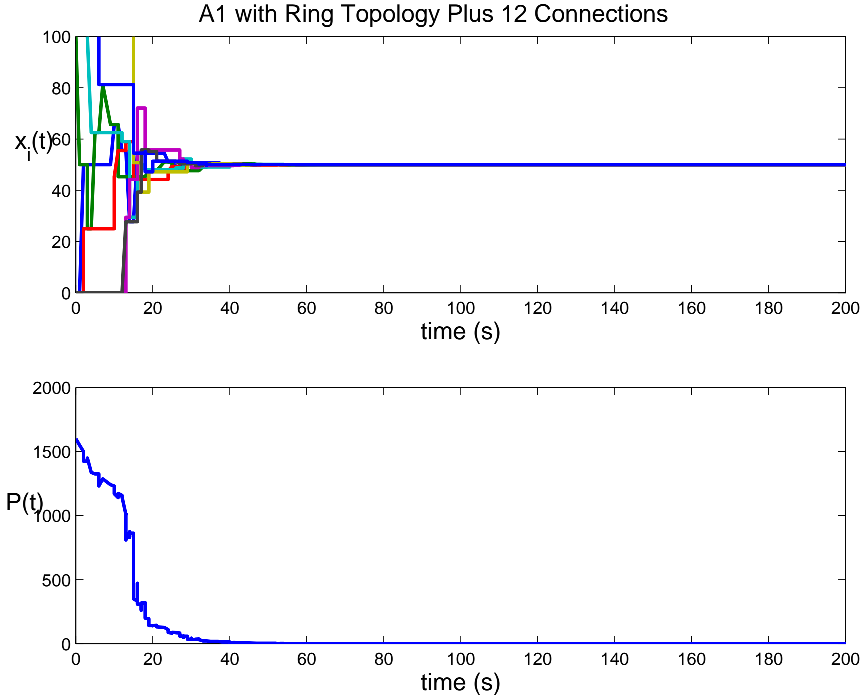


Figure 2.9: Simulation results of A1 on a ring topology plus 12 extra connections. On the top graph, each color represents a trajectory of one state value over time. The behavior of the potential function over time is plotted on the bottom.

reciprocal of the number of nodes. Each node can therefore have a running estimate of how large the active network size is without any additional action from the bootstrapping server.

2.5 Algorithm A2

The blocking behavior for algorithm A1 requires occasional dropping of packets, which may not be desirable when node power is a scarce resource. Moreover, it constitutes most of the coding complexity in the implementation of A1. As an alternative, we will propose another algorithm, denoted by A2.

In A2, each node i makes use of the additional variables δ_{ij} as described in section 2.4. As described earlier, if there is a link between nodes i and j, there will be variables δ_{ij} and δ_{ji} stored locally with node i and node j, respectively.

We will denote the set of all neighbors of node i to be N_i. The algorithm A2 is specified mathematically in terms of the x_i's and the δ_{ij}'s as follows in the synchronous environment:

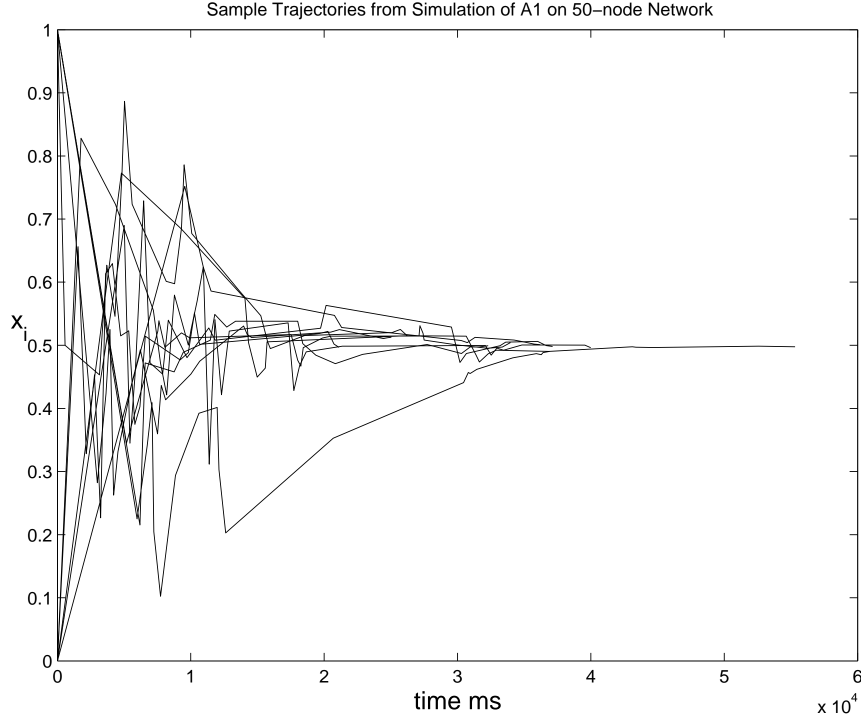


Figure 2.10: State histories from a simulation of algorithm A1 on a fifty-node network. Round-trip delays on each link were assigned randomly, between 40 (ms) and 1000 (ms). Note that all states converge towards the average value .5.

$$\begin{cases} x_i(t+1) = x_i(t) + \gamma_i \left[\sum_{j \in N_i} \delta_{ij}(t) + z_i - x_i(t) \right], \\ \delta_{ij}(t+1) = \delta_{ij}(t) + \phi_{ij} \left[x_j(t) - x_i(t) \right], \end{cases} \tag{2.13}$$

where we introduce the additional parameters ϕ_{ij}, which are local stepsizes similar to γ_i.

Algorithmically, the above update rules require additional specifications. First of all, each x_i is initialized to z_i as in algorithm A1, and each δ_{ij} is initialized to 0. If there is a link between i and j, the parameters ϕ_{ij} and ϕ_{ji} are set to be equal. (We will see that one can also just set all ϕ's on the network to some constant value.)

Second, in order to guarantee convergence to the correct average, we require the following messaging rules. On each link ij, we impose a *sender-receiver* relationship on the variables δ_{ij} and δ_{ji}. One can use UIDs to obtain this, as described in section 2.3.2.

MR1: Every node i sends to every neighbor a STATE message that contains its current state

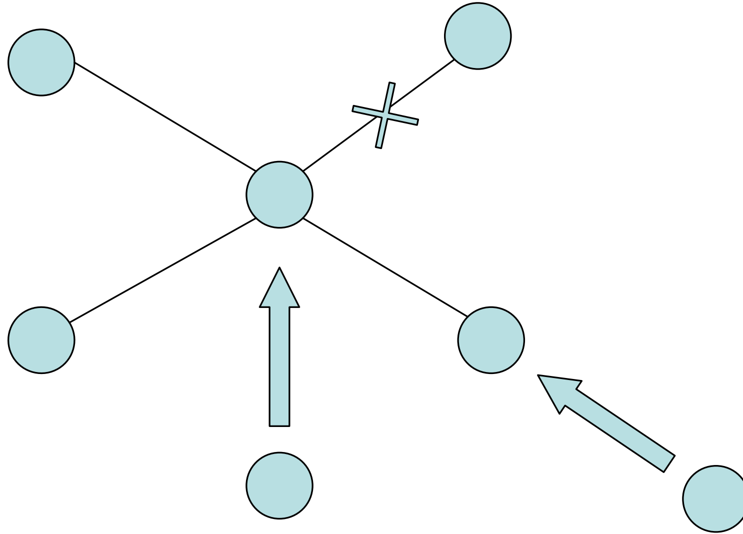


Figure 2.11: An illustration of the joins and leaves of peers on a peer-to-peer network. Existing peers can leave the system either voluntarily or due to failure. New peers can join the network by connecting to any of the existing peers.

value x_i from time to time. Each node also, from time to time, executes the update rule (first equation in (2.13)) with the information it has about other state values.

MR2: On link ij, if δ_{ij} is the sender, it executes the update rule (second equation in (2.13)) from time to time. Whenever δ_{ij} executes the update rule, it also sends to its receiver δ_{ji} a REPLY message that contains the value of the *c*hange it has made in the value of δ_{ij}. δ_{ij} will not execute the update rule again until the TCP ACK of this REPLY message comes back.

MR3: If δ_{ji} is the receiver on link ij, it waits for REPLY messages from δ_{ij} and subtracts the value in the message from the value of δ_{ji}. (Note that the REPLY message does not directly change the value of x_j.)

Notice that the second equation in (2.13) is general enough to cover the execution required in MR3. Also, since the δ_{ij} variables are employed, A2 is automatically equipped with the ability to handle dynamic topologies (as discussed in Section 2.4). All node i needs to do is to reset δ_{ij} to 0 if node j leaves the system.

We can now obtain the following property for δ_{ij}, which is important for A2's convergence to the correct average.

Lemma 4 *For any pair of variables δ_{ij} and δ_{ji}, at any time t, there exists a later time $t' > t$ such that*

$$\delta_{ij}(t') + \delta_{ji}(t') = 0. \tag{2.14}$$

Proof 4 *Initially all δ_{ij}'s are set to 0. According to the messaging rules MR2 and MR3, if δ_{ij} executes the update rule (second equation in (2.13)) and changes its value by some amount at time t, the opposite change will be made by δ_{ji} at some later time t'. Therefore their sum becomes 0 again.*

If we consider the vector consisting of all the values of x_i and δ_{ij}, equations (2.13) can be thought of as an affine mapping on the vector of states. We will show that the mapping is a contraction mapping, provided the stepsizes γ_i and ϕ_{ij} satisfy

$$\begin{cases} 0 < \gamma_i < \frac{1}{d_i+1}, \\ 0 < \phi_{ij} < \frac{1}{2}. \end{cases}$$

Notice that the stepsize constraints are local: Each node only needs to know the local degree d_i to determine the above stepsize bounds.

In the convergence proof of A1, we have used the fact that there are no overlapping updates on adjacent links. Therefore, we can ignore the message-passing details and just consider each complete pairwise update in a sequence of discrete time instants as in (2.2). A2 does not impose any blocking constraint and thus it does not have this property for simple analysis. In particular, under A2, after node i sends out a state message to node j, node i is allowed to immediately send out another state message to some other neighbor k regardless of when node j's reply arrives. While waiting for a reply from node j, node i can also accept any incoming reply messages, and any state messages from all neighbors (including node j).

In order to prove convergence of A2, we will make use of a general and powerful framework of asynchronous computation in [6]. We enumerate all these message-passing events in the set $T = \{0, 1, 2, ...\}$, and let $T^{ij} \subset T$ be the set of times when δ_{ij} updates its value,

and $T^i \subset T$ be the set of times when x_i updates its value. Equations (2.13) become

$$\begin{cases} x_i(t+1) = x_i(t) + \gamma_i \left[\sum_{j \in N_i} \delta_{ij}(\tau_{ij}^i) + z_i - x_i(t) \right] \text{ if } t \in T^i, \\ x_i(t+1) = x_i(t), \text{if } t \notin T^i, \\ \delta_{ij}(t+1) = \delta_{ij}(t) + \phi_{ij} \left[x_j(\tau_j^{ij}) - x_i(\tau_i^{ij}) \right] \text{ if } t \in T^{ij}, \\ \delta_{ij}(t+1) = \delta_{ij}(t), \text{ if } t \notin T^{ij}, \end{cases}$$

where $0 \leq \tau_i^{ij}, \tau_j^{ij}, \tau_{ij}^i \leq t$ indicate possibly "old" copies of the variables involved in the update equations. (See [6] for more details.)

It can be shown that the following asynchronous timing assumption guarantees convergence of all the states to the desired average value:

Total asynchronism: (as defined in [6]) Given any time t_1, there exists a later time $t_2 > t_1$ such that

$$\tau_i^{ij}(t) \geq t_1, \tau_{ij}^i(t) \geq t_1, \forall i, j, \text{ and } t \geq t_2. \tag{2.15}$$

This is in spirit similar to the Eventual Update Assumption for A1. In general, we have the following asynchronous convergence theorem for A2:

Theorem 2

$$\lim_{t \to \infty} x_i(t) = \frac{1}{n} \sum_{i=1}^{n} z_i, \forall i$$

under A2 with total asynchronism, provided the stepsizes satisfy

$$\begin{cases} 0 < \gamma_i < \frac{1}{d_i+1}, \\ 0 < \phi_{ij} < \frac{1}{2}. \end{cases} \tag{2.16}$$

Proof 5 *It can be shown that given the stepsize constraints, the synchronous equations are a* contraction mapping *with respect to the infinity norm. To illustrate this, consider a simple*

case with a two-node network. The synchronous update equations are

$$\begin{cases} x_1(t+1) = x_1(t) + \gamma_1 \left[\delta_{12}(t) + z_1 - x_1(t)\right], \\ x_2(t+1) = x_2(t) + \gamma_2 \left[\delta_{21}(t) + z_2 - x_2(t)\right], \\ \delta_{12}(t+1) = \delta_{12}(t) + \phi_{12} \left[x_2(t) - x_1(t)\right], \\ \delta_{21}(t+1) = \delta_{21}(t) + \phi_{21} \left[x_1(t) - x_2(t)\right]. \end{cases} \tag{2.17}$$

The linear part of the mapping is therefore

$$\begin{pmatrix} 1-\gamma_1 & 0 & \gamma_1 & 0 \\ 0 & 1-\gamma_2 & 0 & \gamma_2 \\ -\phi_{12} & \phi_{12} & 1 & 0 \\ \phi_{21} & -\phi_{21} & 0 & 1 \end{pmatrix}. \tag{2.18}$$

If the stepsize bounds (2.16) are satisfied, this linear mapping is strictly diagonally dominant (namely, the magnitude of every diagonal entry is strictly larger than the sum of the magnitudes of all the other entries in its row.) In the general case, the x_i row has a diagonal entry of $1-\gamma_i$, d_i entries of value γ_i, and entries of value 0 otherwise. The δ_{ij} row has a diagonal entry of 1, an entry of value ϕ_{ij}, an entry of value $-\phi_{ij}$, and 0 otherwise. Therefore it can be seen that diagonal dominance is true in general, provided (2.16) is satisfied. Using Proposition 2.1 of Section 6.2 in [6], A2 converges under total asynchronism.

Now denote x_i^*, δ_{ij}^* to be the limit points of x_i and δ_{ij}. We see from the update equations that

$$\begin{cases} x_i^* = z_i + \sum_{j \in N_i} \delta_{ij}^*, \forall i, \\ x_i^* = x_j^*, \forall i, j. \end{cases}$$

Therefore,

$$\begin{aligned} x_i^* &= \frac{1}{n}\sum_{i=1}^{n} x_i^* \\ &= \frac{1}{n}\sum_{i=1}^{n} z_i + \frac{1}{n}\sum_{i=1}^{n}\sum_{j\in N_i}\delta_{ij}^* \\ &= \frac{1}{n}\sum_{i=1}^{n} z_i, \forall i, \end{aligned}$$

where in the last step we use Lemma 4 to cancel out all the δ_{ij}^* *terms.*

2.5.1 Convergence Rate

Analytical characterization of the convergence rate of our algorithms is difficult to obtain due to their general assumptions. In the case of algorithm A1, the potential function presented in section 2.3.1 can serve as a metric for convergence rate. It is shown in the proof of Theorem 1 that this potential function decreases exponentially in time.

For algorithm A2, however, we do not have any analytical results for the convergence rate in general. It is worth noting that the convergence rate of the synchronous algorithm (2.1) (as described in section 2.2) can be characterized analytically. It is shown in [34] that the convergence rate is related to the second-smallest eigenvalue of the Laplacian matrix. This value is also known as the Fiedler eigenvalue or the algebraic connectivity of the graph. We have observed empirically that the convergence rate of the asynchronous algorithm A2 is similar to that of the synchronous algorithm (2.1) with the same average delays.

2.5.2 Experimental Results

We developed an implementation of A2 in a C socket program and deployed it on the PlanetLab network [35]. The PlanetLab network is a collaborative research network that supports large-scale networking experiments. We have chosen some long-range topologies that span across the globe for the experiments. An example snapshot of the PlanetLab network is illustrated in Figure 2.12.

We performed several runs of the algorithm, each time randomly choosing 50 to 100 nodes. Round-trip delays on this network ranged between tens of milliseconds and one

Planetlab Testbed

Figure 2.12: A snapshot of the PlanetLab world-wide research network. Our experiments were carried out on overlay networks of these nodes.

second. Various overlay topologies were tested, with consistent convergence on the order of a few seconds.

In each experimental run, every node obtained a list of neighbors from a central server and established TCP connections to its neighbors. After the topology-formation phase was completed, the nodes were each sent a message instructing them to begin the iterative computation with their neighbors. One sample of these experimental results is shown in Figure 2.13.

In this experiment, an overlay network of 100 nodes on Planetlab was chosen. Half of the nodes started with initial value $z_i = 0$, and the other half with $z_i = 1$; therefore the target average value was .5. Each node was connected to 5 random neighbors in the overlay network. The round-trip times (RTT) on this global overlay network ranged between tens of milliseconds to roughly half a second. It is worth noticing how rapid the convergence is: In 10 seconds all state values converged to within 1% of the average. We believe this is very promising for many applications.

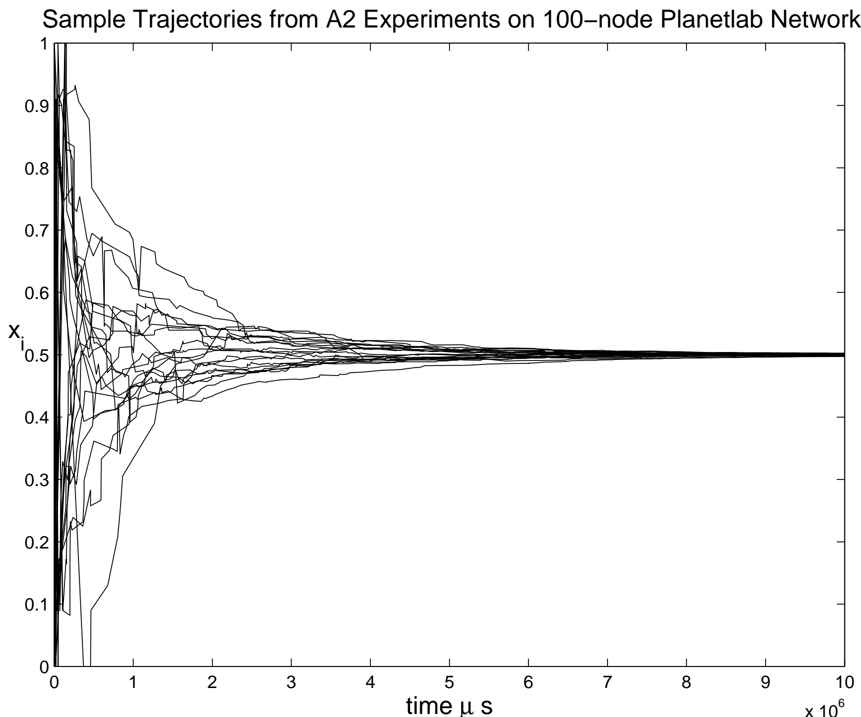


Figure 2.13: Sample histories from an experiment on the PlanetLab network, using 100 nodes and algorithm A2. Round-trip times on this network ranged between tens of milliseconds to approximately one second. Note the rapid convergence of the estimates.

2.5.3 Simulation of A2

Experiments with more than 100 nodes were not feasible for us on the PlanetLab network. Therefore we have written a discrete event simulator in Java and simulated algorithm A2 on larger network sizes.

The network sizes tested were: 50, 100, 250, 500, and 1000 nodes. For each size, the network topology was chosen such that each node connected to 5 randomly selected neighbors. The one-way delay on each link from node i to node j was chosen uniformly randomly between 20 milliseconds and 500 milliseconds. For each network size, there were 5 runs of simulation with the same connectivity but possibly different link delays in each run. All simulation started with half the nodes having initial states 0 and the other half having initial states 1.

In order to see the dependence of the convergence rate on the network size, we kept track of the amount of time it took for all node states to converge to within 1% of the target average. We will call this the "convergence time" for the sake of comparison. The

convergence time (in units of seconds) obtained from our simulation for each run and each network size is reported in Table 2.1.

Table 2.1: Convergence Time v.s. Network Size

Network Size	Run 1	Run 2	Run 3	Run 4	Run 5	Average	Fiedler Eigenvalue
50	7.37	7.34	7.20	7.35	7.01	7.25	1.038
100	7.55	8.42	7.48	7.71	7.69	7.77	1.058
250	8.38	8.41	8.26	7.62	7.96	8.12	1.046
500	8.46	8.08	7.97	8.03	8.40	8.18	1.048
1000	7.81	8.29	8.60	8.39	8.10	8.24	1.020

The convergence time is therefore roughly constant regardless of the size of the network. Notice that this is consistent with our discussion on the convergence rate of the synchronous algorithm in section 2.5.1. The convergence rate in general does not depend on the network size but only on the algebraic connectivity or Fiedler eigenvalue of the topology. The simulation results suggest that our asynchronous algorithms still perform very well when the network size is large.

2.6 Applications and Extensions

2.6.1 Calculating the N-th Moment of Measurement Distributions on Sensor Networks

Instead of just calculating the average of node states, one can also readily adapt the algorithms to calculate the variance or any other moment of the distribution of node states. To get the variance, for example, one simply needs to run another distributed averaging process on the quantities z_i^2. It is therefore straightforward to get the variance given the average of the original values, and the average of the squared values.

A similar approach can be carried out for calculating any n-th moment of the distribution. This may be useful for taking measurements on sensor networks, since the signals can be noisy and additional information other than the average can be desirable.

2.6.2 Tracing Dynamic Network Averages

In addition to calculating the n-th moment of a distribution, we will present a few more extensions of the averaging algorithms.

So far we have assumed that the quantities being averaged (z_i) are static during the executions of our averaging algorithms. It turns out that one can readily adapt the algorithms to handle time-varying local variables, $z_i(t)$ (details can be found in [43]). All one need do is to modify the local state x_i each time z_i changes. Specifically, every time the local variable changes, according to

$$z_i \leftarrow z_i + \Delta z,$$

then the local state is modified according to

$$x_i \leftarrow x_i + \Delta z.$$

This ensures that at any given time, the sum of the node states is equal to the sum of the local variables. Each time one of the z_i changes, the iterative algorithm adapts and converges to the appropriate value.

2.6.3 Dynamic Node Counting in a Transient Peer-to-peer Network

As mentioned earlier, one promising application of distributed averaging is a mechanism for counting the number of nodes on a peer-to-peer network. Due to the transient nature of peer-to-peer networks, it is often hard to obtain a good estimate of the total number of active nodes.

Suppose it can be ensured that one and only one node has set $z_i = 1$, while all others have set $z_i = 0$. The averaging algorithms will therefore converge to a target value of $1/n$, the reciprocal of the number of nodes. If *all* nodes were completely identical, this coordination would be hard to achieve. This is possible, however, since peer-to-peer systems usually have a small centralized component. For example, the bootstrapping server on a Kazaa network, or the tracker of a BitTorrent network can be the special node with $z_i = 1$.

An example of a network of nodes and their values for the counting application is shown in Figure 2.14. Notice that one and only one of the nodes has value 1 and the other nodes have value 0.

With this arrangement of the input values z_i, each node can therefore use distributed averaging to obtain a running estimate of how large the active network size is, without any

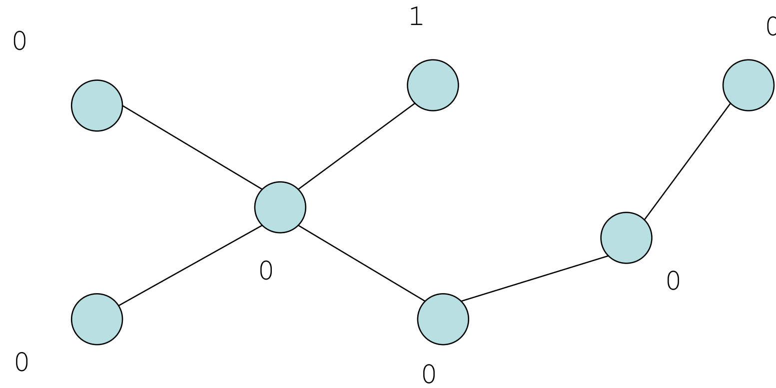


Figure 2.14: An example network for the counting application of distributed averaging. Notice that one and only one peer has an initial value of 1, and the remaining peers have a value of 0. The average value of this particular network is 1/7, the reciprocal of the number of peers.

additional action from the bootstrapping server or the tracker.

2.6.4 Improving Peer Selection Algorithms in BitTorrent

Peer-to-peer file sharing will be discussed in more detail later. We would like to point out here, however, that distributed averaging can conceivably be used to improve the peer selection algorithms in BitTorrent.

BitTorrent peers use a "tit-for-tat" algorithm for deciding what peers to allocate upload capacities to. The idea is to create an incentive for sharing, and also to favor peers with large capacities. With the tit-for-tat algorithm, each peer keeps a rolling estimate of the download rates received from its neighbors, and then it reciprocates by allocating its upload capacity to the peers who have given it the best download rates. Notice that peers can only measure the download rates from its neighbors, and therefore they can only try to find the best peers *locally*.

Another algorithm BitTorrent uses for deciding what peers to allocate capacities to is the "optimistic unchoking" algorithm. Besides the tit-for-tat algorithm, each BitTorrent peer uploads to a randomly selected peer who may not have had any reciprocal contributions. This is to make sure that every peer has a chance to receive some data even if they are very slow, and to probe the potential capability of other choices of peers who may actually be faster than the current choices.

Each BitTorrent peer typically uploads to four peers by tit-for-tat and one peer by optimistic unchoking. The four-to-one ratio is not adaptive. Distributed averaging can help the tit-for-tat algorithm by providing the peers with more information about the rates of the whole system. It is possible for each peer to keep track of the average of the upload capacities of the *whole system*, and therefore choose to tune the ratio. For example, if a peer finds that the rates it is receiving from its current tit-for-tat peers are not very good in comparison with the global average rate, the peer can decide to "probe" more and carry out more optimistic unchoking.

2.7 Summary and Conclusion

We have presented a class of practically implementable distributed averaging algorithms that are suitable for communication networks such as the Internet. Our algorithms do not rely on synchronization, knowledge of the global topology, or coordination of parameter values.

Our analytical results for A1 show that under a mild timing assumption, the asynchronous message-passing algorithms can achieve exponential convergence. For the case when the blocking behavior of A1 is undesirable, we have introduced algorithm A2, which is free of the blocking requirement and is also provably convergent under very general asynchronous timing. The iterative nature of the algorithm renders it robust to changes in topology.

We have presented simulations, as well as experimental results from a real-world TCP/IP network. These results demonstrate the desired convergence behavior and show that the algorithms proposed can be implemented robustly in a practical network. We further show that the convergence speed does not degrade as the size of the network grows. The algorithms are therefore very promising for real-world applications.

Chapter 3

Modelling BitTorrent-like Peer-to-peer File Sharing

3.1 Introduction to Peer-to-peer File Sharing

3.1.1 Demand for Large Content

There is a growing demand for obtaining large content through the Internet. As the hardware for storage of information has become cheaper, the networking infrastructure for the distribution of information has become relatively expensive. In order to distribute large content, a traditional content distribution infrastructure can be hard to scale.

In the traditional client-server setting, clients divide up the download capacity of the server, and as the number of clients grows, the server quickly becomes a performance limiting bottleneck. The main idea behind peer-to-peer file sharing is to make use of the capabilities of the *clients*.

According to Web analysis firm CacheLogic, peer-to-peer traffic accounted for an astounding 60% of all traffic on the Internet at the beginning of year 2005. BitTorrent accounted for half of that, namely, 30%.

3.1.2 The BitTorrent Protocol

The basic idea of BitTorrent is to divide the content into small file pieces, so that peers can help upload file pieces they have already obtained to other peers in the network. The typical size file piece is 256KB, a quarter megabyte in size.

To start a BitTorrent file sharing network, a "torrent" file is made available for peers to obtain, usually on the Web. The torrent file contains information about the file such as

its name, file size, and IP address of the "tracker." A tracker is a centralized component of BitTorrent that helps peers find other peers in the network to connect to. In order to verify data integrity, a hash value of every file piece is included in the torrent file.

When a new peer obtains the torrent file and tries to join the network, it first connects to the tracker. The tracker responds with a list of IPs of peers who are downloading the same file in the network, for the new peer to try to connect to and become neighbors with.

Typically the tracker responds with a random list of less than 50 peers. When a new peer is connected to a list of other peers in the system, it receives information about what pieces its neighbors have. It also reports to its neighbors what pieces it has, whenever it receives new pieces.

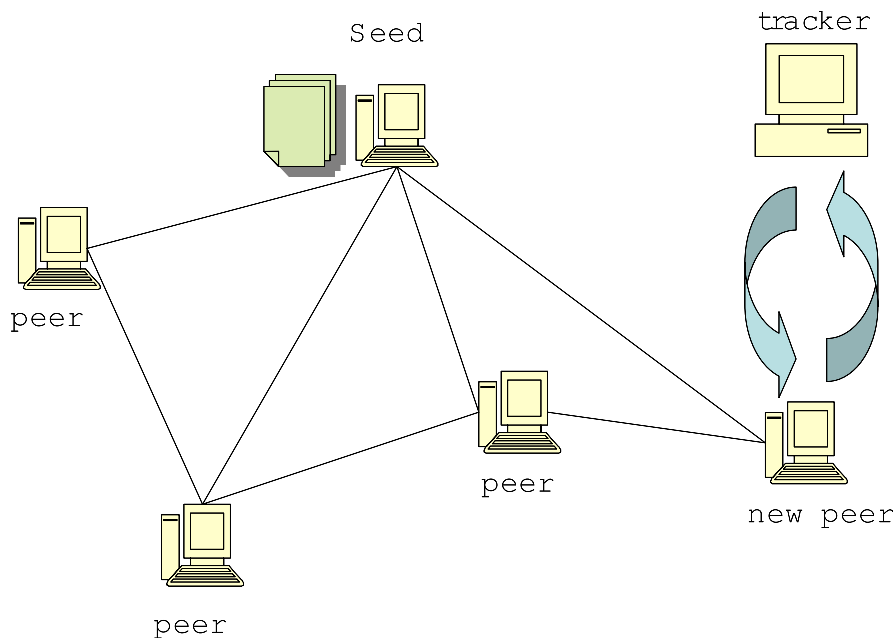


Figure 3.1: A schematic for the BitTorrent protocol. New peers first obtain a torrent file, and then ask the tracker for IP addresses of existing peers to connect to.

BitTorrent peers use a "tit-for-tat" algorithm for deciding what peers to allocate upload capacities to. The idea is to create incentive for sharing, and also to favor peers with large capacities. With the tit-for-tat algorithm, each peer keeps a rolling estimate of the download rates received from its neighbors, and then it reciprocates by allocating its upload capacity to the peers who have given it the best download rates.

Another algorithm BitTorrent uses for deciding what peers to allocate capacities to

is called "optimistic unchoking." Besides the tit-for-tat algorithm, each BitTorrent peer uploads to a randomly selected peer who may not have had any reciprocal contributions. This is to make sure that every peer has a chance to receive some data even if they are very slow.

Another important part of the BitTorrent protocol is the piece selection algorithm. After deciding what peers to upload to, a peer also needs to decide what pieces to send. BitTorrent uses a "rarest-first" policy for piece selection. A peer chooses to upload the file piece that is the rarest (by looking at what has what pieces among its neighbors) to the receiving peer. This ensures that pieces can spread out quickly to the entire network.

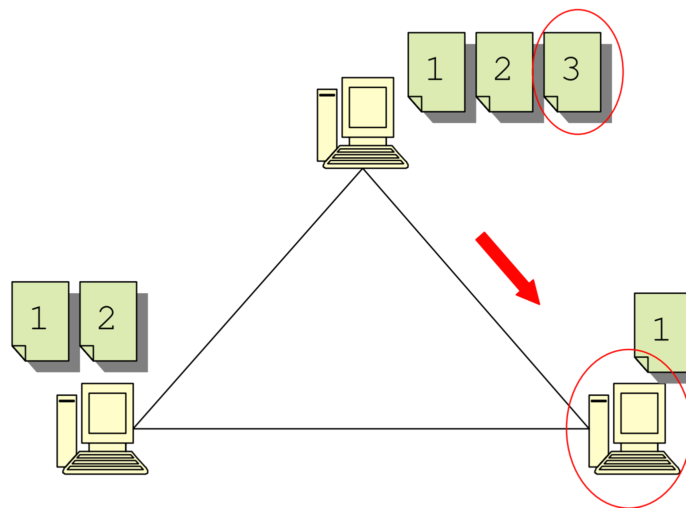


Figure 3.2: A schematic for the rarest-first policy employed by the BitTorrent protocol. Pieces that are "rare" in the network have a higher chance of being uploaded than pieces that are commonly found among peers. Here, the top peer chooses to send piece 3 instead of piece 2 to the peer on the bottom right, because piece 3 is less common in the local network than piece 2.

We will see that the file-sharing model we study in this work captures many of these important underlining principles.

3.2 Model Setup

Consider a network of N peers where each peer's goal is to obtain the same content, a file of size F. The file is divided into P pieces of equal size to facilitate distribution. When a

peer has received a piece completely, it can help distribute the piece by sending it to other peers. This method of file dissemination is used by file-sharing systems such as BitTorrent [10], Slurpie [41], and Avalanche [15].

In addition to the N peers who initially have no file pieces, we assume there is a server that initially has the whole file. This server is called a seed node in BitTorrent. The upload capacity of each peer is assumed to be the only constraint, which is an assumption motivated by the fact that peers usually have larger download capacity than upload (e.g., DSL lines) on the Internet. We also assume that the overlay network is a complete graph, and therefore there is no connectivity constraint.

We are interested in the time it takes for each peer to obtain the whole file. We denote by t_i the *finish time* of peer i, which is defined to be the earliest time at which peer i receives the whole file. The values of t_i depend on the *strategy* by which the system decides how to allocate capacity and distribute data among peers. Although it is possible to provide a strict definition of a strategy, we will not attempt to do so since the complexity of such formulation may not be worthwhile.

When P, the number of pieces that the file is divided into, is very large, the data packets can be considered as continuous flows. In this work, we focus on the case where P is infinite, and therefore data can be considered as continuous fluid.

The notation of this model is as follows:

F: size of the file

N: total number of peers (not including the server)

C_s: upload capacity of the server

C_i: upload capacity of peer $i, \forall i = 1, 2, ..., N$

$F_i(t)$: amount of file that peer i has at time t

Notice that by definition, we must have

$$0 \leq F_i(t) \leq F, \forall t. \tag{3.1}$$

Without loss of generality, we assume that the peer capacities are in decreasing order,

namely,

$$C_i \leq C_j, \forall i > j. \tag{3.2}$$

We also define the total capacity of the system to be

$$C := C_s + \sum_{i=1}^{N} C_i. \tag{3.3}$$

We will also call the time

$$\frac{F}{C_s} \tag{3.4}$$

to be the *bottleneck time* of the system, since it is the least amount of time the server needs to upload file F to any peer.

Notice that in this model, the analysis is not simply about allocating capacities. Peers can only upload to other peers when they have actually received the data in the first place. Also, the receiving peers must *not* have received such data. The analysis, therefore, has to take into account how different file segments are distributed, in addition to how capacities are allocated.

3.3 Last Finish Time

We will call the amount of time for all peers to obtain the file the *last finish time*. This is referred to as the "minimum makespan" in [32]. In other words, the last finish time T_L is defined to be

$$T_L := \max_i \{t_i\}, \tag{3.5}$$

the time when the last peer finishes.

When the server and only the server has the file initially, it can be shown [33] that the minimal last finish time T_L^* is given by the following simple expression.

Theorem 3 *The minimal last finish time is given by*

$$T_L^* = \max\left\{\frac{F}{C_s}, \frac{NF}{C}\right\}. \tag{3.6}$$

First we notice that both terms in the maximization must be lower bounds of T_L^*, since F/C_s is the least amount of time the server needs to upload the file, and NF/C is the least amount of time for the whole system (with combined capacity $C = C_s + \sum_j C_j$) to upload N copies of the file needed. Therefore, the above theorem holds as long as the equality can be achieved. Mundinger et al. [32] discovered the following strategy, which we will denote by S_0:

(i) When

$$C_s \geq \frac{\sum_i C_i}{N-1}, \tag{3.7}$$

it is possible for the server to allocate to each peer i an upload rate of

$$\frac{C_i}{N-1}. \tag{3.8}$$

Peer i can therefore upload to *all* the other $N-1$ peers at this rate, without exceeding its capacity constraint C_i, because

$$\frac{C_i}{N-1}(N-1) \leq C_i. \tag{3.9}$$

Now notice that the server still has

$$C_s - \frac{\sum_i C_i}{N-1} \tag{3.10}$$

amount of capacity to upload. This will be shared among all the N peers. Therefore, the capacity each peer i receives is equal to

$$\frac{C_s - \frac{\sum_{j=1}^N C_j}{N-1}}{N} + \sum_{j=1}^{N} \frac{C_j}{N-1} = \frac{C}{N}, \forall i = 1, 2, ..., N. \tag{3.11}$$

(ii) When

$$C_s < \frac{\sum_i C_i}{N-1}, \tag{3.12}$$

the server can allocate to each peer i an upload rate of

$$\frac{C_i C_s}{\sum_{j=1}^{N} C_j}, \tag{3.13}$$

without exceeding its capacity constraint C_s. Peer i can therefore upload to *all* the other $N-1$ peers at the rate of

$$\frac{C_i C_s}{\sum_{j=1}^{N} C_j} \tag{3.14}$$

without exceeding its capacity constraint C_i because

$$(N-1)\frac{C_i C_s}{\sum_{j=1}^{N} C_j} < C_i. \tag{3.15}$$

Therefore, the capacity each peer i receives is equal to

$$\sum_{i=1}^{N} \frac{C_i C_s}{\sum_{j=1}^{N} C_j} = C_s, \forall i = 1, 2, ..., N. \tag{3.16}$$

Under the above scheme (S_0), it is easy to see that not only can each peer receive a high enough rate, it can also receive *distinct* file segments from the server and the other peers. Therefore, strategy S_0 indeed gives all peers the whole file F, at time T_L^*. In addition, all peers finish at the same time under S_0, i.e.,

$$t_i = T_L^*, \forall i = 1, 2, ..., N. \tag{3.17}$$

We argue in this work that the last finish time may not be the most suitable objective of interest for peer-to-peer file-sharing systems. Notice that the expression of T_L^* suggests that peers with small capacity joining the system can have an *arbitrarily large* impact on system performance. This is certainly not true in a BitTorrent network, for example. In fact, since strategy S_0 is such that all peers finish at the same time (perfect fairness), the

efficiency of the system is compromised.

Lemma 5 *When $N = 2$, strategy S_0 is always Pareto-optimal.*

Proof 6 *To show that S_0 is Pareto-optimal when $N = 2$, we need to show that it is* not *possible to reduce one finish time of S_0 without increasing the other. From (3.6) we see that the values of T_L^* have only two cases:*

(i) When

$$t_1 = t_2 = F/C_s, \tag{3.18}$$

the finish times are clearly Pareto-optimal since it is not possible to reduce either of the finish times.

(ii) When

$$t_1 = t_2 = 2F/C, \tag{3.19}$$

suppose another strategy changes t_1 to some value t_1' with $t_1' < 2F/C$, and t_2 to some value t_2'. We argue that we must have $t_2' > 2F/C$. Since $t_1' < 2F/C$, during the time period $[t_1', 2F/C]$, peer 2 cannot upload to any peer since peer 1 has already finished. Therefore, the whole system cannot upload at full capacity C for the entire time period $[0, t_2']$. This implies $t_2' > 2F/C$. The same argument applies when the indices 1 and 2 are exchanged.

Thus we know S_0 is always Pareto-optimal when $N = 2$.

Strategy S_0 is in general not Pareto-optimal, however. We will state the following result but skip the proof:

Lemma 6 *When $N \geq 3$ and $T_L^* > F/C_s$, strategy S_0 is not Pareto-optimal.*

3.4 Other Optimality Criteria

3.4.1 Average Finish Time

In this work, we propose to study objectives other than the last finish time. We argue that in the context of peer-to-peer file sharing, the last finish time may not be as important as

the *average finish time*, for example. The average finish time T_A is defined to be the average of all finish times:

$$T_A := \frac{1}{N} \sum_{i=1}^{N} t_i. \tag{3.20}$$

A simple example would illustrate why the average finish time may be more of interest to the peers than the last finish time. Consider the special case where all peer capacities are 0. If server capacity C_s is split equally among all peers, every peer will finish at the same time

$$t_i = \frac{NF}{C_s}, \forall i = 1, 2, ..., N. \tag{3.21}$$

However, if the downloads are scheduled separately in time, the finish times can be

$$t_i = \frac{iF}{C_s}, \forall i = 1, 2, ..., N. \tag{3.22}$$

Therefore, the average finish time in (3.22) becomes

$$\frac{(N+1)\, F}{2C_s}, \tag{3.23}$$

nearly half of the average value of the finish times in (3.21). The overall user experience is undoubtedly better in (3.22), and it can be shown that the finish times in (3.22) are actually *optimal* for average finish time.

To see that the finish times in (3.22) are optimal, we consider the following inequalities. Since by time t_i the whole system must have at least uploaded i copies of the file F, we have

$$t_i \geq \frac{iF}{C}, \forall i = 1, 2, ..., N. \tag{3.24}$$

Since all the peer capacities are 0, this is equivalent to

$$t_i \geq \frac{iF}{C_s}, \forall i = 1, 2, ..., N. \tag{3.25}$$

We thus see that

$$T_A = \frac{1}{N}\sum_{i=1}^{N} t_i \tag{3.26}$$

$$\geq \frac{(N+1)\,F}{2C_s}. \tag{3.27}$$

The finish times in (3.22) achieve the above lower bound, and therefore are optimal for average finish time T_A.

3.4.2 Min-Min Finish Times

It is also possible to consider the following "min-min" optimality criterion for the finish times, where each finish time t_i is sequentially minimized, in order from fast to slow peers.

$$t_1^m := \min t_1 = \frac{F}{C_s}, \tag{3.28}$$

$$t_i^m := \min \left\{ t_i | t_j = t_j^m, \forall j < i \right\}. \tag{3.29}$$

In other words, the min-min finish time t_i^m of peer i is the minimal possible value of t_i subject to the constraints that the finish time of any peer j with an index $j < i$ is equal to $t_j = t_j^m$.

3.5 General Properties

We will first show that during any time period where no peer finishes, we can assume without loss of generality that no file segment is uploaded by more than two nodes. In other words, no file segment traverses more than two "hops" when it is sent.

Suppose a file segment B starts from node i_0 (either a peer or the server) to peers i_1, i_2 and i_3 in sequence, as shown in Figure 3.4. We claim that this data flow can be replaced by a union of several two-hop data flows without changing the finish times of any peer.

First, we divide the file segment B into two disjoint parts B_1 and B_2 of equal size. Then node i_0 sends B_1 to peer i_1, and B_2 to peer i_2. When peer i_1 receives B_1, it broadcasts B_1 to the other two peers. Similarly, when peer i_2 receives B_2, it broadcasts B_2 to the the other two peers as well. This is illustrated in Figure 3.3. Notice that peers i_1 and i_2 use the same capacity to upload since $2|B_1| = 2|B_2| = |B|$. Therefore, peers receive the same file

segments with the modified flow of Figure 3.3 as in Figure 3.4. It is easy to apply the same argument to any multi-hop flows. Therefore we see that it is without loss of generality to assume data flows that traverse at most two hops.

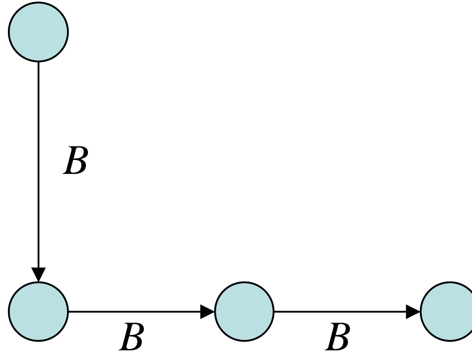


Figure 3.3: A file segment B is uploaded by three nodes in this graph. It can be regarded as the data traversing three hops in the flow.

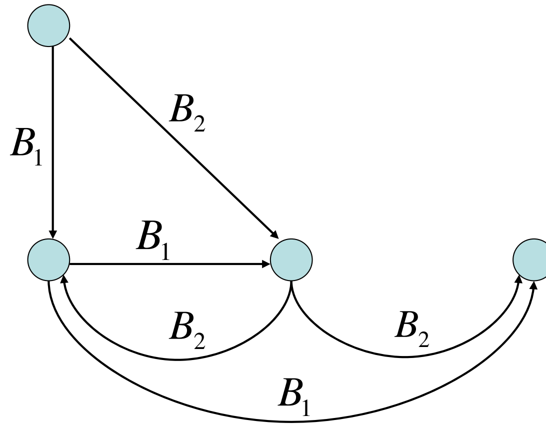


Figure 3.4: A file segment is divided into two disjoint segments of equal size, B_1 and B_2. Notice that each sending node still uses the same upload capacity, and each receiving node still receives the same segments. This flow is now "two-hop."

We will now present another general property of the model.

Theorem 4 *(Multiplicity Theorem) It is possible to let the first M peers finish at bottleneck time if and only if*

$$C_s \leq \sum_{i=1}^{M} \frac{C_i}{M-1} + \sum_{i=M+1}^{N} \frac{C_i}{M}. \tag{3.30}$$

Proof 7 *(i) Consider the set of all peers $i \in \{1, 2, ..., M\}$ as set A_1, and the rest of the peers $i \in \{M+1, M+2, ..., N\}$ as set A_2. We will derive an upper bound on the total amount of data set A_1 as a whole can receive. It is clear that any data A_1 receives must be from either the server, other peers in A_1, or peers in set A_2. The server and peers in A_1 can at most upload at the rate of*

$$C_s + \sum_{i=1}^{M} C_i. \tag{3.31}$$

Each node j in set A_2 can broadcast any data that it receives to all the peers in A_1. The server can therefore route through peer j and "multiply" its sending rate to set A_1 by a factor of M, subject to the upload constraint of peer j. The cost is, of course, that this will inevitably leave one copy of the data with peer j, which is not in set A_1. The net contribution of routing through peer j to A_1 is therefore a factor of $M-1$. The most peer j can receive in this situation is C_j/M since otherwise it does not have enough upload capacity to broadcast to M peers. The net contribution from A_2 to A_1 is therefore at most

$$\sum_{i=M+1}^{N} \frac{M-1}{M} C_i. \tag{3.32}$$

Now, if it is possible to finish all peers A_1 at bottleneck time, then the total amount of data A_1 receives must be at least as much as M copies of the file, namely, MF. Therefore, we have

$$MF \leq \left(C_s + \sum_{i=1}^{M} C_i + \sum_{i=M+1}^{N} \frac{M-1}{M} C_i \right) \frac{F}{C_s}, \tag{3.33}$$

and this is equivalent to

$$C_s \leq \sum_{i=1}^{M} \frac{C_i}{M-1} + \sum_{i=M+1}^{N} \frac{C_i}{M}. \tag{3.34}$$

(ii) Suppose inequality (3.30) holds. We want to show that it is possible to let the first M

peers finish at bottleneck time. We can write the server capacity as

$$C_s = \lambda \left(\sum_{i=1}^{M} \frac{C_i}{M-1} + \sum_{i=M+1}^{N} \frac{C_i}{M} \right), \tag{3.35}$$

where $0 < \lambda \leq 1$.

Consider the following strategy:

(i) The server uploads to each peer in set A_1 *at rate*

$$\frac{\lambda C_i}{M-1}, \tag{3.36}$$

and each peer broadcasts the data it receives from the server to any other peer in set A_1.

(ii) The server uploads to each peer in set A_2 *at rate*

$$\frac{\lambda C_i}{M}, \tag{3.37}$$

and each peer broadcasts the data it receives from the server to all peers in set A_1.

It is straightforward to check that each peer in set A_1 *receives a total data rate of* C_s*, with distinct file segments from each of the senders. Therefore, all peers in set* A_1 *will finish at bottleneck time.*

Given the server and peer capacities, we will define the *multiplicity* of the system to be the largest M such that inequality (3.30) holds. In other words, the multiplicity is M if it is possible to finish the first M peers at bottleneck time, but not the first $M + 1$ peers.

Note that the multiplicity of any system is at least 1, since it is always possible to finish peer 1 at bottleneck time. This fact can also be seen from the right-hand side of (3.30), as it becomes infinite when $M = 1$ and, therefore, the inequality is always true for any finite C_s.

One important consequence of the Multiplicity Theorem (Theorem 4) is that it suggests a natural decomposition of our model into different cases according to the relative magnitude of server capacity versus peer capacities. More specifically, when there are N peers in the

network, there will be N different intervals the server capacity C_s can be in, and the model exhibits different behaviors in those intervals. We will illustrate these behaviors with some examples.

3.6 Optimal Average Finish Time

Suppose the objective now is to minimize the average finish time

$$T_A = \frac{1}{N} \sum_{i=1}^{N} t_i. \tag{3.38}$$

We will show in this section that in many cases, the optimal average can be achieved by the min-min finish times defined in Section 3.4.2.

3.6.1 Optimal Average when $M = N$

First of all we note that it is obvious how to obtain the optimal average finish time when the multiplicity M is equal to N; namely, when

$$C_s \leq \frac{\sum_{i=1}^{N} C_i}{N-1}. \tag{3.39}$$

It is easy to see that since

$$t_i \geq \frac{F}{C_s}, \forall i = 1, 2, ..., N, \tag{3.40}$$

the optimal average finish time must satisfy

$$T_A^* \geq \frac{F}{C_s}. \tag{3.41}$$

Now, when $M = N$, the equality in (3.41) is actually achieved by S_0. Therefore,

$$T_A^* = \frac{F}{C_s}, \text{ if } M = N, \tag{3.42}$$

and it is easy to see the finish times of all peers are just the bottleneck time F/C_s, which are equal to the min-min finish times in this case as well.

Intuitively one can think of this as the case with the server being the "bottleneck," and thus the the finish times only involve the capacity of the server.

3.6.2 Network of Two Peers

The simple case with one server and two peers ($N = 2$) can only have two possible values for multiplicity, 1 and 2. When $M = 2$, according to the previous analysis, the optimal finish times are just simply the bottleneck times.

When $M = 1$, following similar analysis we can see the following:

Lemma 7 *The optimal average finish time for $N = 2$ and $M = 1$ is*

$$\frac{F}{C_s} + \frac{F\,(C_s - C_1 - C_2)}{2C_s\,(C_s + C_1)}, \tag{3.43}$$

with the peers finishing at times

$$\frac{F}{C_s}, \text{ and } \frac{F}{C_s} + \frac{F\,(C_s - C_1 - C_2)}{C_s\,(C_s + C_1)}. \tag{3.44}$$

In addition, the optimal average finish time is strictly better than the optimal last finish time

$$T_A^* < T_S^*. \tag{3.45}$$

Proof 8 *If $t_1 \leq t_2$, at time t_1, we have*

$$F_1(t_1) + F_2(t_1) \leq (C_s + C_1 + C_2)t_1, \tag{3.46}$$

because the sum of the files that the peers have obtained must be upper bounded by the largest possible total upload of the whole system. Thus we have

$$F_2(t_1) \leq (C_s + C_1 + C_2)t_1 - F, \tag{3.47}$$

where equality is achieved if all capacities are utilized during $[0, t_1]$. After peer 1 finishes,

peer 2 *can obtain data from both the server and peer* 1*, and thus*

$$t_2 = t_1 + \frac{F - F_2(t_1)}{C_s + C_1} \tag{3.48}$$

$$\geq t_1 + \frac{2F - Ct_1}{C_s + C_1}. \tag{3.49}$$

The average finish time therefore satisfies

$$T_A = \frac{t_1 + t_2}{2} \tag{3.50}$$

$$\geq t_1 + \frac{2F - Ct_1}{2\left(C_s + C_1\right)} \tag{3.51}$$

$$\geq \frac{F}{C_s} + \frac{F\left(C_s - C_1 - C_2\right)}{2C_s\left(C_s + C_1\right)}. \tag{3.52}$$

If $t_1 \geq t_2$*, the same argument gives*

$$T_A \geq \frac{F}{C_s} + \frac{F\left(C_s - C_1 - C_2\right)}{2C_s\left(C_s + C_2\right)} \tag{3.53}$$

$$\geq \frac{F}{C_s} + \frac{F\left(C_s - C_1 - C_2\right)}{2C_s\left(C_s + C_1\right)}. \tag{3.54}$$

Therefore, inequality (3.52) holds in both cases. Consider the following strategy:

(i) During $[0, t_1]$*, server uploads different file segments to peers* 1 *and* 2 *at the rates of* $C_s - C_2$ *and* C_2*, respectively. Peer* 1 *forwards the data it receives from the server to peer* 2 *at the rate of* C_1*. Peer* 2 *forwards its receiving data at rate* C_2 *to peer* 1*. Peer* 1 *therefore receives a combined rate of* C_s*, and finishes at time* $t_1 = F/C_s$*. Peer* 2 *receives a combined rate of* $C_1 + C_2$*, which is less than* C_s*, and therefore it does not finish at time* F/C_s*.*

(i) During $[t_1, t_2]$*, the server and peer* 1 *combines their capacities to upload to peer* 2 *the remaining file that peer* 2 *still needs. Peer* 2 *therefore finishes at*

$$t_2 = t_1 + \frac{F - (C_1 + C_2)\frac{F}{C_s}}{C_s + C_1} \tag{3.55}$$

$$= \frac{F}{C_s} + \frac{F}{C_s}\left(\frac{C_s - C_1 - C_2}{C_s + C_1}\right). \tag{3.56}$$

It can be checked that this strategy achieves the equality in (3.52), and hence the first

part of this lemma is proved. Now consider

$$T_A^* - T_S^* = \frac{F}{C_s} + \frac{F(C_s - C_1 - C_2)}{2C_s(C_s + C_1)} - \frac{2F}{C_s + C_1 + C_2} \tag{3.57}$$

$$< \frac{F}{C_s} + \frac{F(C_s - C_1 - C_2)}{2C_s(C_s + C_1)} - \frac{2F}{2C_s} \tag{3.58}$$

$$= \frac{F(C_s - C_1 - C_2)}{2C_s(C_s + C_1)} \tag{3.59}$$

$$< 0. \tag{3.60}$$

Therefore, the second part of the lemma is proved too.

3.6.3 Optimal Average when $M = N - 1$

It turns out that the analysis for $N = 2$ and $M = 1$ can readily be extended to any N and $M = N - 1$. From the Multiplicity Theorem, we know that $M = N - 1$ is equivalent to

$$\frac{\sum_{i=1}^{N} C_i}{N-1} < C_s \leq \frac{\sum_{i=1}^{N-1} C_i}{N-2} + \frac{C_N}{N-1}. \tag{3.61}$$

Therefore, we can always write C_s in the form of

$$C_s = \lambda \left(\sum_{i=1}^{N-1} C_i \right) + \frac{C_N}{N-1}, \tag{3.62}$$

where λ is a constant such that

$$\frac{1}{N-1} < \lambda \leq \frac{1}{N-2}. \tag{3.63}$$

Consider the following strategy:

(i) The server uploads different file segments to peers i, $i = 1, 2, ..., N - 1$, at the rate of

$$\lambda C_i. \tag{3.64}$$

Each peer then broadcasts the data it receives from the server to any other peer i in $\{1, 2, ..., N - 1\}$. Peer i still has some remaining capacity equaling

$$C_i - (N-2)\lambda C_i, \tag{3.65}$$

so it can upload to peer N at this remaining rate.

(ii) The server uploads to peer N at rate

$$\frac{C_N}{N-1}, \tag{3.66}$$

and peer N broadcasts the data it receives from the server to all other peers in the network. Note that the upload capacity of the server is saturated, and so is the upload capacity of peer N.

We will now show that the above strategy is optimal for average finish time. First we note that at time $t = t_{N-1}$ we must have

$$(N-1)\,F + F_N(t_{N-1}) \leq Ct_{N-1}, \tag{3.67}$$

since Ct_{N-1} is the most amount of total uploads from the entire system, and the left hand side is the amount of data all peers have in total at time $t = t_{N-1}$. Furthermore, it is easy to see that

$$t_N - t_{N-1} \geq \frac{F - F_N(t_{N-1})}{C - C_N}, \tag{3.68}$$

because, during time $[t_{N-1}, t_N]$, the server and all peers other than peer N itself can upload to peer N. Now from inequality (3.67) we know that

$$F - F_N(t_{N-1}) \geq NF - Ct_{N-1}. \tag{3.69}$$

Therefore (3.68) becomes

$$t_N - t_{N-1} \geq \frac{NF - Ct_{N-1}}{C - C_N}. \tag{3.70}$$

Using (3.70) we see that

$$\begin{aligned} NT_A &= \sum_{i=1}^{N} t_i && (3.71)\\ &= \sum_{i=1}^{N-2} t_i + 2t_{N-1} + (t_N - t_{N-1}) && (3.72)\\ &\geq \sum_{i=1}^{N-2} t_i + 2t_{N-1} + \frac{NF - Ct_{N-1}}{C - C_N} && (3.73)\\ &= \sum_{i=1}^{N-2} t_i + \frac{NF}{C - C_N} + \frac{C - 2C_N}{C - C_N} t_{N-1} && (3.74)\\ &\geq \frac{(N-2)F}{C_s} + \frac{NF}{C - C_N} + \frac{C - 2C_N}{C - C_N}\frac{F}{C_s}. && (3.75) \end{aligned}$$

In the last inequality (3.75) above, we use the fact that

$$t_i \geq \frac{F}{C_s}, \forall i, \quad (3.76)$$

and that

$$\frac{C - 2C_N}{C - C_N} \geq 0. \quad (3.77)$$

It can be shown that the strategy we propose results in finish times

$$\begin{aligned} t_i &= \frac{F}{C_s}, \forall i = 1, 2, ..., N-1, && (3.78)\\ t_N &= \frac{F}{C_s}\frac{NC_s - C_N}{C - C_N}, && (3.79) \end{aligned}$$

which *achieve* the equality in (3.70). Therefore, it is optimal for average finish time.

3.6.4 Networks of Three Peers

When $N = 3$, the only case we have not shown a strategy for the optimal average is the case when $M = 1$. This is the case when

$$C_s > C_1 + C_2 + \frac{C_3}{2}. \quad (3.80)$$

We will show that the following strategy is optimal for average finish time:

Theorem 5 *When $N = 3$, $M = 1$, the following strategy is optimal in average finish time:*

(i) During time period $[0, t_1]$, the server sends different file segments to peers 1, 2, and 3 at rates $C_s - C_2 - r_3$, C_2, and r_3, respectively, where

$$r_3 := \frac{2C_s - C_2}{2C_s + 2C_1 + C_3} C_3. \tag{3.81}$$

It can be shown that

$$\frac{C_3}{2} \leq r_3 \leq \min\{C_3, C_s - C_1 - C_2\}, \tag{3.82}$$

therefore, it is possible for the server to allocate the rate of r_3 to peer 3. Then peer 1 uploads to peer 2 at rate C_1; peer 2 uploads to peer 1 at rate C_2; peer 3 uploads to peer 1 at rate r_3, and to peer 2 at rate $C_3 - r_3$.

(ii) During time period $[t_1, t_2]$, peer 3 continues to upload to peer 2 the data it received from the server during $[0, t_1]$. The server and peer 1 each uploads at its full rate to peer 2, and peer 2 uploads at its full rate to peer 3.

(iii) During time period $[t_2, t_3]$, the server and peers 1 and 2 upload to peer 3 at a combined rate of $C_s + C_1 + C_2$, and finish peer 3.

Proof 9 *First of all, we show that (3.82) is true, and therefore it is possible for the server to allocate r_3. First we have*

$$\begin{aligned} r_3 - \frac{C_3}{2} &= \frac{C_3}{2C_s + 2C_1 + C_3}\left(2C_s - C_2 - \frac{2C_s + 2C_1 + C_3}{2}\right) \\ &= \frac{C_3}{2C_s + 2C_1 + C_3}\left(C_s - C_1 - C_2 - \frac{C_3}{2}\right) \\ &> 0. \end{aligned}$$

Also, we have

$$\begin{aligned} \min\{C_3, C_s - C_1 - C_2\} - r_3 &= \min\{C_3 - r_3, C_s - C_1 - C_2 - r_3\} \\ &= \min\left\{\frac{2C_1 + C_2 + C_3}{2C_s + 2C_1 + C_3} C_3, \frac{(C_s + C_1)(2C_s - 2C_1 - 2C_2 - C_3)}{2C_s + 2C_1 + C_3}\right\} \\ &\geq 0. \end{aligned}$$

Therefore, it is possible for the server to allocate the rate of r_3 *to peer* 3.

Let the set of the peers $i \in \{1, 2\}$ *be set* A*; then an upper bound of the total amount of data that can go into set* A *at time* t_2 *is*

$$C_s t_2 + C_1 t_2 + C_2 t_1 + \frac{C_3}{2} t_2. \tag{3.83}$$

The reason for the first three terms in (3.83) is that the server and peer 1 *can potentially upload data to set* A *at full capacity during* $[0, t_2]$*, and peer* 2 *can only upload to set* A *during* $[0, t_1]$*, since during* $[t_1, t_2]$ *there is no destination in set* A *for peer* 2 *to upload to anymore. Since peer* 3 *can at most upload the same data to two peers, the net contribution from peer* 3 *to set* A *is at most*

$$\min\{C_3 t_2, 2F_3(t_2)\} - F_3(t_2), \tag{3.84}$$

which is at most

$$\frac{C_3}{2} t_2. \tag{3.85}$$

Now, since we know that at time $t = t_2$*, peers* 1 *and* 2 *have both finished, the most possible data that can go into set* A *must be at least* $2F$*. Therefore, we have*

$$C_s t_2 + C_1 t_2 + C_2 t_1 + \frac{C_3}{2} t_2 \geq 2F, \tag{3.86}$$

and it is equivalent to

$$t_2 \geq \frac{2F - C_2 t_1}{C_s + C_1 + \frac{C_3}{2}}. \tag{3.87}$$

Adding t_1 *to both sides of (3.87), and adding* $t_1 + t_2$ *to both sides of (3.70) with* $N = 3$

gives

$$
\begin{aligned}
t_1 &\geq \frac{F}{C_s}, && (3.88)\\
\sum_{i=1}^{2} t_i &\geq \frac{2F}{C_s + C_1 + \frac{C_3}{2}} + \frac{C_s + C_1 + \frac{C_3}{2} - C_2}{C_s + C_1 + \frac{C_3}{2}} t_1, && (3.89)\\
\sum_{i=1}^{3} t_i &\geq \frac{3F}{C - C_3} + \frac{C_3 t_1}{C - C_3} + \frac{C - 2C_3}{C - C_3} \left(t_1 + t_2\right). && (3.90)
\end{aligned}
$$

It is then possible to substitute for t_1 and $t_1 + t_2$ in (3.90) and obtain a lower bound on $t_1 + t_2 + t_3$. It can therefore be seen that if a strategy achieves equality in each of the three inequalities (3.88-3.90), it has to be optimal for average finish time. One can check that the strategy we propose indeed achieves all three equalities, and therefore it is optimal for average finish time.

We illustrate numerically the optimal finish times for average in Figure 3.5. The parameters are chosen to be $C_s = 500$, $C_1 = 200$, $C_2 = 80$, $C_3 = 70$, and $F = 100$. Each of the finish times t_1, t_2, t_3 is plotted as a bar with its height equaling its value. Also plotted are the values of T_A^* and T_L^*. We see that the optimal average finish time T_A^* is significantly better than the optimal last finish time T_L^*.

We illustrate the behavior of the finish times that optimize the average according to the three different multiplicity values in Figure 3.6.

3.6.5 Networks of Four and More Peers

It is possible to generalize (3.86) to the case of general N. For each index j such that $2 \leq j \leq N-1$, consider the set A of peers $\{1, 2, ...j\}$. An upper bound on the amount of data that can go into set A at time t_j is

$$\left(C_s + \sum_{i=1}^{j-1} C_i\right) t_j + C_j t_{j-1} + \frac{j-1}{j} \left(\sum_{i=j+1}^{N} C_i\right) t_j. \quad (3.91)$$

Since all peers in set A finish at or before t_j,

$$\left(C_s + \sum_{i=1}^{j-1} C_i\right) t_j + C_j t_{j-1} + \frac{j-1}{j} \left(\sum_{i=j+1}^{N} C_i\right) t_j \geq jF, \forall N, j, 2 \leq j \leq N-1. \quad (3.92)$$

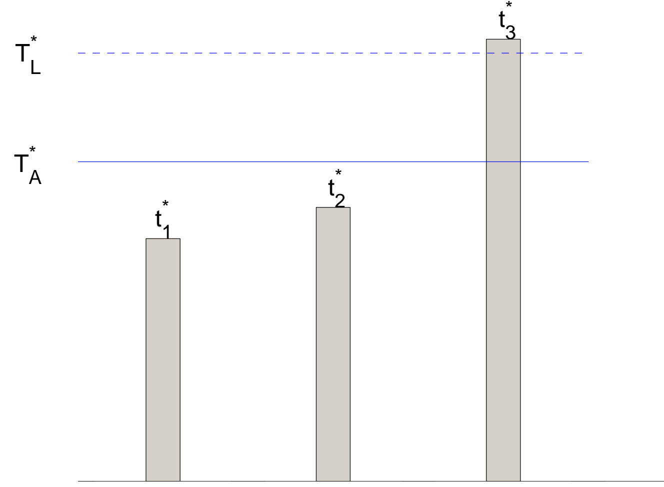


Figure 3.5: An illustration of the finish times that are optimal for average. The parameters chosen are $C_s = 500$, $C_1 = 200$, $C_2 = 80$, and $C_3 = 70$. The optimal values of T_A and T_L are also plotted.

It is equivalent to

$$t_j \geq \frac{jF - C_j t_{j-1}}{C_s + \sum_{i=1}^{j-1} C_i + \frac{j-1}{j} \sum_{i=j+1}^{N} C_i}, \forall N, j, 2 \leq j \leq N-1. \tag{3.93}$$

When $N = 4$, from the Multiplicity Theorem we know that there are four different intervals for C_s corresponding to four different multiplicities:

$$\begin{cases} M = 4, & \text{if } 0 \leq C_s \leq \frac{C_1+C_2+C_3+C_4}{3}, \\ M = 3, & \text{if } \frac{C_1+C_2+C_3+C_4}{3} < C_s \leq \frac{C_1+C_2+C_3}{2} + \frac{C_4}{3}, \\ M = 2, & \text{if } \frac{C_1+C_2+C_3}{2} + \frac{C_4}{3} < C_s \leq C_1 + C_2 + \frac{C_3+C_4}{2}, \\ M = 1, & \text{if } C_s > C_1 + C_2 + \frac{C_3+C_4}{2}. \end{cases} \tag{3.94}$$

If we could find a strategy that achieves equality in all of the inequalities in (3.93), by the same argument used in the $N = 3$ case, we would show that the finish times of such a strategy are average optimal, and are the min-min finish times at the same time.

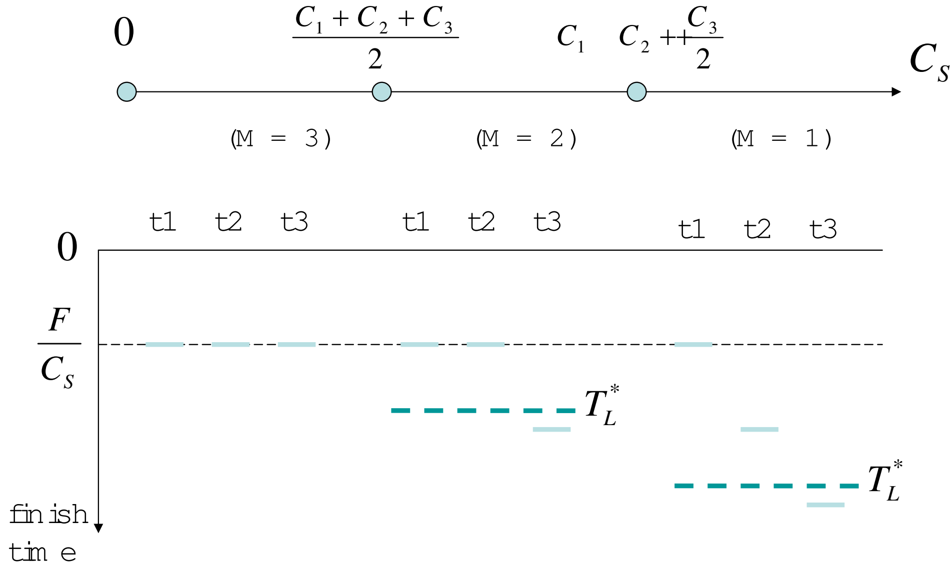


Figure 3.6: An illustration of the behavior of the finish times that are optimal for average, with different multiplicities. There are three intervals C_s can be in, and they are illustrated on top. The intervals are marked with their corresponding value of multiplicity. An illustration of the different behaviors for different intervals is on the bottom. Notice how the peer finish times "spread out" as C_s increases. The optimal value of the last finish time, T_L^*, is also drawn for comparison.

Furthermore, we would have a closed-form expression for each of the finish times.

However, it turns out that there does not always exist a strategy that achieves all of the inequalities in (3.93). We conjecture that min-min finish times are always average optimal in general, but the behavior of the optimal t_i can be more complex than the expression in (3.93).

3.7 Min-min Finish Times

As mentioned in Section 3.4.2, it is possible to consider sequential minimization of finish times in order of decreasing capacity as an optimality criterion. The min-min finish times are defined to be

$$t_1^m := \min t_1 = \frac{F}{C_s}, \tag{3.95}$$

$$t_i^m := \min \left\{ t_i | t_j = t_j^m, \forall j < i \right\}. \tag{3.96}$$

In other words, each peer's finish time is minimized subject to the constraint that the finish times of all lower-indexed peers are minimized.

From the Multiplicity Theorem (Theorem 4), it is easy to see the following:

Lemma 8 *If the multiplicity is M, the min-min finish times are such that*

$$t_i^m = \frac{F}{C_s}, \forall i = 1, 2, ..., M, \tag{3.97}$$

$$t_i^m > \frac{F}{C_s}, \forall i, M < i \leq N. \tag{3.98}$$

Proof 10 *The multiplicity is M if and only if it is possible to finish the first M peers at time F/C_s, therefore the first M min-min finish times have to be F/C_s, and the other min-min finish times cannot be F/C_s.*

We observe that min-min finish times are closely related to the finish times that achieve the optimal average. We further conjecture that the problem of minimizing the average is actually equivalent to the sequential minimization process for min-min finish times.

Theorem 6 *The optimal average finish time is achieved by min-min finish times, for all N when M is N or $N-1$, and for all M when $N=3$.*

Proof 11 *We have shown in Section 3.4.1 that min-min finish times are also average optimal when the multiplicity M is equal to N or $N-1$.*

For the case of $N = 3$ and $M = 1$, consider the inequalities (3.88-3.90). One can see that the problem of minimizing $t_1 + t_2 + t_3$ can be reduced to the problem of minimizing $t_1 + t_2$ and t_1. The problem of minimizing $t_1 + t_2$ is similarly reduced to the problem of minimizing t_1. Since the strategy we propose achieves equality in (3.88-3.90), we argue that its finish times must be min-min finish times.

When all equalities are achieved, we see from (3.88) that $t_1 = t_1^m$. Now, by letting $t_1 = t_1^m$, achieving equality in (3.89) indicates that t_2 achieves the minimal value subject to the constraint that $t_1 = t_1^m$. This is by definition the value of t_2^m. Therefore, we see that $t_2 = t_2^m$ as well. A similar argument can be applied to (3.90), and we see that $t_3 = t_3^m$. Therefore, the finish times are min-min finish times.

3.8 Selfish Peers

It is possible to consider peer-to-peer file sharing models that are slightly modified. We will consider a special alternative model that assumes that peers will *leave* the system when they finish, instead of staying to help upload to unfinished peers. So far we have assumed peer altruism in the model, and the alternative model is motivated by the fact that in real peer-to-peer networks, peers are not always altruistic. We would like to point out that in practice, BitTorrent peers have generally exhibited an amazing amount of altruism. This phenomenon is studied in [17].

To capture the case where peers are not altruistic, we can consider an alternative model where we assume peers do *not* upload to other peers once they finish; namely, peer i does not upload to any peer for any time $t > t_i$.

In the non-altruistic case, first we see that the optimal last finish time T_L^* is *not* changed.

Lemma 9 *In the case where peers are selfish, the optimal last finish time is still the same value as the case with altruistic peers:*

$$T_L^* = \max\left\{\frac{F}{C_s}, \frac{NF}{C}\right\}. \tag{3.99}$$

Proof 12 *If the peers are selfish, at any point in time their upload rates can only be less than or equal to the case where they are altruistic. Therefore, C is still an upper bound on the total capacity of the system. It can therefore be seen that F/C_s and NF/C are still lower bounds of T_L^*. Since S_0 still achieves equality for these lower bounds, we see that S_0 is optimal for T_L and hence the lemma is proved, either the peers are selfish or altruistic.*

If we look at the average finish time T_A, however, we see some interesting behavior. To optimize T_A, we can see the following result for the simple $N = 2$ case:

Lemma 10 *If peers are selfish and $N = 2$, then there are two sets of finish times that optimize the average finish time when $M = 1$. One is to finish the faster peer first*

$$\begin{cases} t_1 &= \frac{F}{C_s}, \\ t_2 &= \frac{F}{C_s}\left(\frac{2C_s - C_1 - C_2}{C_s}\right), \end{cases} \tag{3.100}$$

and the other is to finish the slower peer first

$$\begin{cases} t_1 & = \frac{F}{C_s}\left(\frac{2C_s - C_1 - C_2}{C_s}\right), \\ t_2 & = \frac{F}{C_s}. \end{cases} \tag{3.101}$$

They are both optimal for T_A. When $M = 2$, the optimal finish times for average are simply

$$\begin{cases} t_1 & = \frac{F}{C_s}, \\ t_2 & = \frac{F}{C_s}, \end{cases} \tag{3.102}$$

and they are achieved by S_0.

Proof 13 *The $M = 2$ case is trivial. When $M = 1$, we know from the Multiplicity Theorem that*

$$C_s \geq C_1 + C_2. \tag{3.103}$$

Suppose we have $t_2 \geq t_1$, then at time t_1 we have

$$F + F_2(t_1) \leq Ct_1, \tag{3.104}$$

namely,

$$F - F_2(t_1) \geq 2F - Ct_1. \tag{3.105}$$

Now, since the peers are selfish, after peer 1 finishes only the server can upload to peer 2. Therefore, we see that

$$t_2 - t_1 \geq \frac{F - F_2(t_1)}{C_s} \tag{3.106}$$

$$\geq \frac{2F - Ct_1}{C_s}. \tag{3.107}$$

This gives

$$t_1 + t_2 \geq \frac{2F + (2C_s - C)\,t_1}{C_s}. \tag{3.108}$$

Since we know that $2C_s - C \geq 0$ from (3.103), using the fact that $t_1 \geq F/C_s$ we get

$$T_A \geq \frac{F}{2C_s}\left(\frac{3C_s - C_1 - C_2}{C_s}\right). \tag{3.109}$$

It is easy to check that both (3.100) and (3.101) achieve the above lower bound on T_A, and therefore they are both optimal for average finish time.

For larger networks of selfish peers, the general behavior is unknown. Min-min finish times in this case may not be average optimal. Its general behavior is an open problem we are still investigating.

3.9 Summary and Conclusion

We study a model for peer-to-peer file sharing with respect to different optimality criteria. We have derived general properties of the system, and analyzed special cases of system behavior.

Intuitively, the results suggest that efficient peer-to-peer file sharing should have two components. One is that file segments should be spread out to as many peers as possible, in order to utilize every peer's upload capacity. The other component is that fast peers should be favored over slow peers, but they should be not be favored *exclusively*. It seems that fast peers should receive more file segments (but not all file segments) earlier, while at the same time the capacities of slow peers should also be utilized.

BitTorrent's choking algorithms [10] are similar in spirit. There are two major algorithms for deciding which peer to upload to in BitTorrent. One is to reciprocate to the peers who have sent data at the highest rates to you, and the other is to randomly choose a peer (who is possibly very slow) and upload to it.

We have studied Pareto optimality, average optimality, and min-min finish times in this work. There are conceivably numerous other optimality criteria that can provide insights to the understanding of peer-to-peer file sharing systems. We have also considered a special alternative model where peers are assumed to be selfish, or non-altruistic.

It is possible to extend the average finish time T_A to be a weighted average. It is also possible to study more explicitly the tradeoff between fairness and efficiency. Strategy S_0 can be thought of as being "perfectly fair" since it results in equal finish times. To obtain

min-min finish times, or optimize the average finish time, we obtain finish times that favor fast peers over slow peers. The overall performance of the system is improved and is not affected by the presence of possibly numerous slow peers.

It is also possible to consider topology constraints, which are not studied in this work. Also, one may wish to include constraints of download capacities in addition to upload capacities.

Chapter 4

Future Directions

As discussed in Chapter 2, there are many promising applications for distributed averaging. We show that the algorithms are suitable for a variety of different networks in addition to peer-to-peer networks.

The peer-to-peer file sharing model that we have studied in Chapter 2 raises many new open questions. We will conclude the thesis by pointing out a few directions for future work of this model.

4.1 Selfish vs. Altruistic Peers

As studied in Section 3.8, it is possible to consider a modified model where peers are not altruistic. In this "selfish" case, peers finish obtaining the file F and just leave the system. We show that the optimal value for the last finish time T_L^* is not affected by this modification of the model. However, the finish times that optimize the average behave differently in the simple two-peer case.

It seems likely that optimal finish times for average in the selfish case are different from the ones in the altruistic case. Many of the general properties, including the two-hope argument and the Multiplicity Theorem which we have discussed in Chapter 3 still hold, but it is not clear that the optimal finish times would exhibit similar behavior. For instance, it may not be best for fast peers to finish early in the selfish case, since that means fast peers would leave the system early, instead of staying in the system to help.

It would be interesting to study the selfish case more closely and compare it with the altruistic case. We are currently investigating that.

4.2 Data Identity vs. Network Coding

We have not consider the possibility of peers performing computation on the data that they receive. Network coding is a technique that can possibly be employed by peers.

Notice that since many of the performance bounds that we obtain are true regardless of whether network coding is used, it is therefore not possible to *strictly* improve the finish times by using network coding in those cases [9]. For instance, the optimal last finish T_L^* cannot be strictly improved since it already achieves some lower bound that always holds true.

However, it is conceivable that network coding can improve efficiency in some cases. It is also conceivable that network coding can be used to suggest a simpler way to understand the system. So far we have been keeping track of data identity when we talk about strategies for file sharing. It is possible that with network coding, this consideration can be greatly simplified if the data being sent is just generally a linear combination of file segments.

Also, we have not considered the effects of topology constraints. We assume that peers are on a complete graph of connectivity. It is possible that network coding can improve peers' finish times given topology constraints.

4.3 Fairness Objectives

It is possible to think in terms of some measure of "fairness" of the peers' finish times. For example, the objective to minimize the last finish time,

$$\min\left\{\max_i\{t_i\}\right\}, \tag{4.1}$$

can be thought of as minimizing the average finish time, with the fairness constraint that all finish times are the same:

$$\min\left\{\frac{1}{N}\sum_{i=1}^{N}\{t_i\}\right\}, \tag{4.2}$$

$$\text{s.t. } t_i = t_j, \forall i, j. \tag{4.3}$$

It is thus possible to consider different kinds of fairness criteria as constraints to the average-finish-time optimization. An upper bound on the variance of the distribution of the

t_i's, for example, can be a fairness criterion. It can be desirable to consider fairness criteria that favor fast peers over slow peers, in order to reflect the fact that fast peers may upload to the system more than slow peers in general.

Bibliography

[1] E. Adar and B. A. Huberman. Free Riding on Gnutella. *First Monday, Peer-reviewed Journal on the Internet*, vol. 5, no. 10, October 2000. Available at http://firstmonday.org/issues/issue5_10/adar/index.html

[2] R. Ahlswede, N. Cai, S.-Y. R. Li and R. W. Yeung. Network Information Flow. *IEEE Transactions on Information Theory*, IT-46, pp. 1204-1216, 2000.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci. A Survey on Sensor Networks. *IEEE Communications Magazine*, pp. 102-114, August 2002.

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

[5] BearShare. http://www.bearshare.com

[6] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*, Prentice Hall, 1989.

[7] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip Algorithms: Design, Analysis and Applications. *Proceedings of Infocom*, Miami, 2005.

[8] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham and S. Shenker. Making Gnutella-like P2P Systems Scalable. *Proceedings of ACM Sigcomm*, Karlsruhe, Germany, August, 2003.

[9] D. M. Chiu, R. W. Yeung, J. Huang and B. Fan Can Network Coding Help in P2P Networks? Available at http://personal.ie.cuhk.edu.hk/~dmchiu/p2pnetcoding.pdf

[10] B. Cohen. Incentives Build Robustness in BitTorrent, http://bitconjurer.org/BitTorrent/bittorrentecon.pdf

[11] R. Cox, A. Muthitacharoen and R. Morris. Serving DNS using Chord. *First International Workshop on Peer-to-Peer Systems*, Cambridge, USA, March, 2002.

[12] F. Dabek, F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. *Proceedings of ACM Symposium on Operating Systems Principles*, pp. 202-215, Banff, Canada, 2001.

[13] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: scalable coordination in sensor networks. *Proceedings of Mobile Computing and Networking*, 1999.

[14] A. Fax and R. M. Murray. Information Flow and Cooperative Control of Vehicle Formations, *IEEE Transactions on Automatic Control*, vol. 49, pp. 1465-1476, September 2004.

[15] C. Gkantsidis and P. Rodriguez. Network Coding for Large Scale Content Distribution. *Proceedings of IEEE Infocom*, Miami, 2005.

[16] Gnutella. http://www.gnutella.com

[17] D. Hales and S. Patarin. How to cheat BitTorrent and why nobody does. Technical Report UBLCS-2005-12, Department of Computer Science, University of Bologna, May 2004.

[18] S. H. Hong. The Effect of Napster on Recorded Music Sales: Evidence from the Consumer Expenditure Survey. SIEPR Discussion Paper No. 03-18, January, 2004.

[19] M. Jelasity, W. Kowalczyk, and M. van Steen. An approach to massively distributed aggregate computing on peer-to-peer networks. *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 2004.

[20] Kazaa. http://www.kazaa.com

[21] F. Kelly, A. Maulloo and D. Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *Journal of Operations Research Society*, 49(3):237–252, March 1998.

[22] D. Kempe, A. Dobra and J. Gehrke. Computing Aggregate Information using Gossip. *Proceedings of FOCS*, 2003.

[23] D. Kempe and F. McSherry. A Decentralized Algorithm for Spectral Analaysis. *Proceedings of STOC*, 2004.

[24] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells and B. Zhao. OceanStore: An architecture for global-scale persistent storage. *Proceeedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, November 2000, pp. 190V201.

[25] LimeWire. http://www.limewire.com

[26] S. H. Low and D. E. Lapsley. Optimization Flow Control, I: Basic Algorithm and Convergence. *IEEE/ACM Transactions on Networking*, 7(6):861-75, Dec. 1999.

[27] J. Li, J. Stribling, T. M. Gil, R. Morris and M. F. Kaashoek. Comparing the performance of distributed hash tables under churn. *The 3rd International Workshop on Peer-to-Peer Systems*, San Diego, USA, February 2004.

[28] N. Lynch. *Distributed Algorithms*, Morgan Kaufmann Publishers, 1997.

[29] M. Mehyar, D. Spanos, J. Pongsajapan, S. H. Low, and R. M. Murray. Distributed averaging on asynchronous communication networks. *Proceedings of the IEEE Conference on Decision and Control*, Seville, Spain, 2005.

[30] R. Merris. Laplacian Matrices of a Graph: A Survey. *Linear Algebra and its Applications*, 1994.

[31] N. Minar and M. Hedlund. A Network of Peers: Peer-to-peer Models Through the History of the Internet. *Peer-to-peer: Harnessing the Power of Disruptive Technologies*, edited by Andy Oram, O'Reilly, March 2001.

[32] J. Mundinger and R. Weber. Efficient File Dissemination using Peer-to-Peer Technology. Technical Report, Statistical Laboratory Research Reports 2004-01, Cambridge, January 2004.

[33] J. Mundinger, R. R. Weber and G. Weiss. Analysis of Peer-to-Peer File Dissemination amongst Users of Different Upload Capacities. *Performance Evaluation Review*, Performance 2005 Issue.

[34] R. Olfati-Saber and R. Murray. Consensus Problems in Networks of Agents with Switching Topology and Time-Delays, *IEEE Transactions on Automatic Control*, v. 49, no. 9, pp. 1520-1533, September 2004.

[35] PlanetLab. http://www.planet-lab.org

[36] D. Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks. Proceedings of ACM SIGCOMM, Portland, 2004.

[37] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms*, Heidelberg, pp. 329-350, November, 2001.

[38] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker. A Scalable Content-Addressable Network. *Proceedings of ACM Sigcomm*, San Diego, August 2001.

[39] D. S. Scherber and H. C. Papadopoulos. Distributed Computation of Averages Over Ad Hoc Networks. *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 4, pp. 776-787, April 2005.

[40] SETI@home. http://setiathome.ssl.berkeley.edu/

[41] R. Sherwood, R. Braud and B. Bhattacharjee. Slurpie: A Cooperative Bulk Data Transfer Protocol. *Proceedings of IEEE Infocom*, Hong Kong, 2004.

[42] D. Spanos, R. Olfati-Saber and R. M. Murray. Distributed Sensor Fusion Using Dynamic Consensus. *Proceedings of 16th IFAC World Congress*, Prague, 2005.

[43] D. Spanos, R. Olfati-Saber and R. M. Murray. Dynamic Consensus on Mobile Networks. *Proceedings of 16th IFAC World Congress*, Prague, 2005.

[44] D. Spanos, R. Olfati-Saber and R. M. Murray. Distributed Kalman Filtering in Sensor Networks with Quantifiable Performance. *Proceedings of Information Processing in Sensor Networks*, Los Angeles, USA, 2005.

[45] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 2003.

[46] J. N. Tsitsiklis. Problems in decentralized decision making and computation. Ph.D. thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1984. http://web.mit.edu/jnt/www/PhD-84-jnt.pdf

[47] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athans. Distributed Asynchronous Deterministic and Stochastic Gradient Optimization Algorithms. *IEEE Transactions on Automatic Control*, vol. 31, no. 9, pp. 803-812, 1986.

[48] L. Xiao and S. Boyd. Fast Linear Iterations for Distributed Averaging. *Proceedings of the Conference on Decision and Control*, Maui, USA, 2003.

[49] L. Xiao, S. Boyd, and S. Lall. A Scheme for Asynchronous Distributed Sensor Fusion Based on Average Consensus. *Proceedings of Information Processing in Sensor Networks*, Los Angeles, USA, 2005.

[50] X. Yang and G. de Veciana. Service capacity of peer to peer networks. *Proceedings of IEEE Infocom*, Hong Kong, 2004.

[51] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph and J. D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal On Selected Areas In Communications*, vol. 22, no. 1, January, 2004.