UNIVERSITY OF CINCINNATI

Date: 4-Jun-2010

I, Kevin Henkener, hereby submit this original work as part of the requirements for the degree of:
Doctor of Philosophy
in Computer Science & Engineering

It is entitled:
Two-Hop f-Factors and a Fair and Trustworthy P2P Storage Model

Student Signature: Kevin Henkener

This work and its defense approved by:

Committee Chair: Kenneth Berman, PhD

Fred Annexstein, PhD

Jerome Paul, PhD

Yiming Hu, PhD

Dan Ralescu, PhD
Two-Hop $f$-Factors and a Fair and Trustworthy P2P Storage Model

A dissertation submitted to the

Graduate School
of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

in the Department of Computer Science
of the College of Engineering
by

Kevin R. Henkener
June 2010

B.S. University of Cincinnati, 2001, Cincinnati, Ohio

Committee Chair: Kenneth A. Berman, Ph.D.
Abstract

In this dissertation, we present a novel approach to the problem of distributed (peer-to-peer) backup. Our approach requires that data not be transferred more than two-hops from its source and that each peer store exactly the same amount of data as it distributes to be backed up. These two requirements address two important features of any distributed backup solution - trust and fairness.

In a social network, the hop distance requirement means that in the worst case, a peer’s data is backed up in the local storage of a friend of a friend (FoAF). Our assumption is that this offers a higher degree of trust than simply choosing a random peer. We achieve fairness through the requirement that peers store exactly the same amount of data that they distribute for backup. To facilitate this requirement, our approach uses symmetric exchanges of data. This not only supports fairness, but also enhances trust by introducing a vested interest between peers to preserve the data that they are storing.

We call our approach the *fair two-hop exchange scheme*, or FTHES. We show that existing $f$-factor theory and algorithms can be used to compute an FTHES. Then we introduce and prove a fundamental existence theorem which states that an FTHES always exists under two fairly weak conditions. This theorem leads to a linear time sequential algorithm and an efficient distributed algorithm. We also prove a theorem stating that at most $2n - 3$ exchanges are needed to backup all of the data in our scheme and later conjecture that this may actually have a lower bound of $n$. Finally, we present an application of the FTHES in a content management system.
This page is intentionally left blank.
Acknowledgements

I would first like to thank my parents, Randy and Sandy Henkener, for their continuous love and support throughout my life. They have always stood beside me in my personal, professional, and academic careers. Without them, I would not be the person I am today.

Second, I would like to thank my advisor, Dr. Kenneth Berman. He has served in several different roles in my life including instructor, advisor, and most importantly, friend. Dr. Berman was instrumental in obtaining various funding opportunities for me throughout my graduate school tenure. Without his loyal support and dedication to my efforts, this work would not have been possible. I am indebted to him for his time and patience.

Dr. Fred Annexstein, my co-advisor, was also a big part of this work. His patience, advice, support, and friendship have been invaluable. Thank you, Fred!

I would also like to thank my other committee members, Dr. Jerome Paul, Dr. Yiming Hu, and Dr. Dan Ralescu, for their time and patience in serving on my committee.

Last, but not least, I would like to thank my family and many friends that have supported me through the years. Thanks to Chad Yoshikawa and Svetlana Strunjas for grinding through many years of classes and research along side me. Thanks to my college roommates, Scott Kovacs, Rick Burke, and Will Perry, for providing a welcome escape from the daily rigors of graduate student life. Thanks to Mark Sole for opening doors that would have otherwise been locked, for opening my mind to other worlds of possibility, and most importantly, for being a loyal friend. Finally, thanks to the long list of other friends and family that cannot be mentioned here because these acknowledgements need to fit on one page.
# Table of Contents

1 Introduction ........................................ 7
   1.1 Our Model for Distributed P2P Backup ........... 9
   1.2 Our Approach - Fair Two-Hop Exchange Scheme .... 12
      1.2.1 Graph Theoretical Model .................. 14
      1.2.2 Theorems and Results .................... 14

2 Background and Related Research .................... 16
   2.1 P2P ........................................... 17
      2.1.1 P2P File Sharing ........................ 18
      2.1.2 Backup/Archiving ......................... 18
      2.1.3 Trust and Reputation ..................... 21
      2.1.4 Fairness .................................. 21
      2.1.5 Replication and Redundancy ............... 22
   2.2 Graph Theory .................................. 24
      2.2.1 f-Factors (Integer vs 0/1) ................. 24
      2.2.2 k-Hop f-Factors .......................... 26
2.3 Terminology ................................................. 26

3 Fair Two-Hop Exchange Scheme (FTHES) .................................................. 27

3.1 Notation and Definitions ............................................................... 28
3.2 Modeling Fair Two-Hop Exchange Scheme with Two-Hop $f$-
    factors ......................................................................................... 29
3.3 Fundamental Existence Theorem ......................................................... 31
3.4 Near-Fair Exchange Scheme ............................................................ 40

4 Algorithm for Computing an FTHES ......................................................... 43

4.1 Outline ......................................................................................... 44
4.2 Auxiliary Functions ........................................................................... 44
4.3 Exchanges Involving Strict Local Maximas ........................................... 45
    4.3.1 Identify Strict Local Maximas .................................................. 45
    4.3.2 Computing Exchanges ............................................................. 45
4.4 Building an Ascending Forest ............................................................ 48
4.5 Computing Remaining Exchanges ....................................................... 48
    4.5.1 Exchanges Involving Vertices at Depth $\geq 2$ ........................... 49
    4.5.2 Exchanges Involving Vertices at Depth $\leq 2$ ........................... 52
4.6 Complexity Analysis ........................................................................... 53
    4.6.1 SLMS .................................................................................. 53
    4.6.2 Ascending Forest Operations ................................................... 54
    4.6.3 Exchanges in Ascending Forest ................................................ 54

2
5 Distributed Algorithm for Computing an FTHES

5.1 Identify Strict Local Maximas .................. 56
5.2 Remove Strict Local Maximas Within Two Hops ........ 56
5.3 Ascending Forest Generation .................. 58
5.4 Computing Exchanges .................................. 58
5.5 Complexity ............................................. 61
  5.5.1 Messages Involving Strict Local Maximas ........ 61
  5.5.2 Messages Involving Ascending Forest .......... 62
  5.5.3 Messages Involving Remaining Exchanges ....... 62

6 Minimizing FTHES Size ............................ 63

6.1 FTHES Size ............................................. 63
6.2 Expected Number of Strict Local Maximas ............ 67
6.3 Minimum FTHES Size - NP-Hard ..................... 73

7 MyBook: Application of FTHES ....................... 75

7.1 Collaborative Groups .................................. 76
  7.1.1 Structuring Collaborative Groups - MyBook (MBK) . . 77
7.2 Compile, Organize, Visualize - MyBook Graphical User Interface 81
7.3 Proposed P2P Model .................................. 83
  7.3.1 Collaborative Book - CBK ....................... 83
  7.3.2 Peer Interface ..................................... 84
  7.3.3 Content Distribution - FTHES .................. 85
  7.3.4 Content Tracking ................................. 87
8 Conclusions and Future Research

8.1 Future Research

8.2 FTHES: Considering Edge Capacities

8.2.1 Overview and Notation

8.2.2 Generate Tree Covers

8.2.3 Covering $\ell$

8.2.4 Compute Exchanges in $\tau$

8.2.5 Future Work
List of Figures

1.1 An example P2P network with $f(p)$ marked outside each vertex and a
fair exchange scheme given by the equal size exchanges on the edges... 10

1.2 (a) Example star graph $G$; (b) Only one fair exchange is possible... 12

2.1 Example graph $G$ (f-values marked outside each vertex) and corresponding
$f$-factor $\phi(G)$........................................... 25

3.1 From left-to-right. $G, G' = G$ with bivalent vertices inserted. Notice that
in $G'$ we are forced to choose two-hop paths in order to find an $f$-factor
- a two-hop $f$-factor........................................ 30

3.2 $G$.......................................................... 32

3.3 Strict local maximas (marked in double circles) within two hops of one
another. Making the exchange $\varepsilon(1,8) = 2$ reduces vertex 8 to a non-strict
local maxima.................................................. 33

3.4 Ascending forest $A$ built from $G$ and $f'$........................................ 36

3.5 The vertices in $A$ having depth at least 2........................................ 37

3.6 The vertices in $A$ with depth at most 2........................................ 38
6.1 The maximum number of strict local maximas (shown in red) in a graph $G$ for which the neighborhood sufficiency condition holds is $n - 2$.

6.2 Pareto distributed $f$-values. The $x$-axis shows the different $f$-value sizes. The $y$-axis shows the number of peers having a given $f$-value.

6.3 Randomly distributed $f$-values. The $x$-axis shows the different $f$-value sizes. The $y$-axis shows the number of peers having a given $f$-value.

6.4 Average number of strict local maximas for a graph containing 256 vertices with $f$-values assigned via a random distribution. The $x$-axis shows the number of edges. The $y$-axis shows the average number of strict local maximas for each edge count.

6.5 Average number of strict local maximas for a graph containing 256 vertices with $f$-values assigned via a Pareto distribution. The $x$-axis shows the number of edges. The $y$-axis shows the average number of strict local maximas for each edge count.

6.6 Reduction of the sum of subsets problem to the two-hop $f$-factor problem.

7.1 The general structure of an MBK.

7.2 The MyBook GUI - left panel contains the topic hierarchy; top left panel contains a breadcrumb trail of topics in a given path; top right panel contains file references; bottom left panel is a built-in web browser; bottom right panel allows editing of 0-many notes for a topic.

7.3 Proposed peer interface.

8.1 Split a tree cover $t$ such that $p$ belongs to $\text{deg}_G(p) - 1$ more trees.
Chapter 1

Introduction

People and organizations routinely suffer hard disk failures [30] and virus infections that destroy or damage their data. In many cases, the content is not backed up as this task tends to be either an after-thought, or the cost of the backup system and storage space is prohibitively expensive. Several traditional backup methods include manually copying content to external locations (e.g., flash/optical drives, CD, DVD, magnetic tape), using built-in operating system backup tools (e.g., Microsoft Windows Backup [25], [24], Mac OS X Time Machine [2]) and third party tools (Tivoli Storage Manager [15], Carbonite [4], and Mozy [27] to name a few).

Distributed backup is an important application of peer-to-peer (P2P) networks and an attractive alternative to traditional backup measures. P2P networks have gained considerable attention due to their popularity in various applications. These applications include distributed computing, distributed
storage, anonymous web browsing, media streaming, and file sharing. In these networks, a large number of participating computers act as both client and server (servant) and share resources for storage and computing tasks. This servant property has several advantages including the elimination of single servers as points of failures and the potential for compiling huge amounts of computing power and storage capacity.

When compared to traditional backup methods, distributed backup has a major advantage: cost. The cost of the storage and backup utilities can be huge, especially in large organizations. In many cases, the software required for distributed backup systems is free. Academia has produced several such solutions (e.g., Pastiche [10], PeerStore [22], pStore [3]). P2P’s biggest cost advantage is realized in the cost of disk space.

P2P networks often have thousands of connected users at any given time. An estimate by Douceur et. al. [12] shows that hard disks are typically only 53% full. This result indicates that the amount of available disk space in a P2P network is considerable. Consider a relatively small P2P network of only 1000 peers. If we assume that each has an average hard disk size of 100GB with approximately 47GB unused, the P2P network has approximately 4.7TB of space available for backup. This wealth of available storage is a huge advantage of P2P backups given that there exist peers that are willing to cooperate with one another for the global objective of backing up their files.

P2P backup does not come without several issues that need to be addressed. Two of these issues are trust and fairness. Users tend to be uncom-
fordable backing up data, especially sensitive content, in the local storage of other users that they do not know or trust. It is very important that the P2P backup scheme prevent users from reading, modifying, or deleting data that they are storing for other users. Encryption techniques could be implemented so that the data is private to all but the owner. A peer could validate that its data is intact at the backup site by requesting a random sampling of its bits, for example.

The issue of fairness has garnered much research attention in P2P networks. Much of this research is focused on file contribution and the free-riding that ensues when peers download files but do not contribute any. In the context of P2P backup, it is important that the amount of data that a peer distributes for backup correlates to the amount that it stores for others.

1.1 Our Model for Distributed P2P Backup

In our model, we assume an unstructured P2P network made up of peers who have joined the network for the purpose of backing up their data. Let \( p \in P \) be the set of peers in the network, and let \( f(p) \) be the size of the data (measured in common units, e.g., bytes, kilobytes, megabytes) that each \( p \) has to backup. In our model, each \( p \in P \) will also make exactly \( f(p) \) of its unused storage available to backup data from other peers.

Our model requires that pairs of peers exchange data. We use the term exchange scheme to mean a set of exchanges such that each peer has backed
up all \( f(p) \) units of its data. Within this context, we address fairness and trust in P2P backup as follows.

We achieve fairness as well as load balancing by requiring that pairs of peers exchange an equal amount of data. Since it is unlikely that two peers \( p \) and \( q \) will have exactly the same amount of data to backup (i.e., \( f(p) = f(q) \)), we will allow the units to be split into blocks. In doing so, \( p \) and \( q \) can exchange an equal-sized block. We use the term *fair exchange scheme* to mean an exchange scheme where pairs of peers exchange equal size blocks. Figure 1.1 illustrates a fair exchange scheme and the likely need to split each \( f(p) \) into blocks.

Requiring that pairs of peers exchange equal size blocks also enhances trust by introducing a vested interest for each peer to preserve the data that it is storing. We assume that the peers in the system have joined the
network so that they can backup their data. Any peer that misbehaves by either modifying or deleting any unit(s) that it is storing for other peers can be punished. The simplest punishment is for the peers whose units were modified or delete by the misbehaving peer to delete the units that they are storing for the misbehaving peer. Other “retaliations” include the implementation of reputation systems where a misbehaving peer’s reputation would be decreased and peers would be less likely to exchange with that peer.

Trust can further be enhanced by restricting the pairs of peers in each exchange to be adjacent to one another in the network. We assume that adjacent peers are “familiar” with each other. This is especially true in a social network where adjacent peers are considered as friends.

In our model, we try to minimize the number of total number of exchanges needed for a fair exchange scheme. It is likely that the level of trust will decrease as the number of peers that a peer exchanges with increases. More exchanges also results in more overhead required to track and verify data integrity for each peer. In this sense, a peer may not trust that its data is intact.

The fair exchange scheme that we have described thus far addresses the issues of fairness and trust; however, it is likely to be impractical. If we restrict exchanges to be between adjacent peers (i.e., one hop), it is unlikely that all of the data can be backed up. Consider the example shown in Figure 1.2 where each of the $n$ peers has a single block to backup. In this case, only 2 of the $n$ blocks can be exchanged.
1.2 Our Approach - Fair Two-Hop Exchange Scheme

Surprisingly, if we allow exchanges between peers that are at most two hops apart and introduce two somewhat weak conditions, we can always find a fair exchange scheme. Allowing two hops means that data is at most two hops from its owner. In a social network such as Facebook [13], this would mean that in the worst case, a peer would exchange data with a friend of a friend (FoAF). In fact, friend of a friend is an option in Facebook that allows friends of a user’s friends to view that user’s profile. It is assumed that a friend or a friend of a friend is a more trusted backup location than simply choosing a random peer.
Definition 1.1. A *Fair Two-Hop Exchange Scheme* (*FTHES*) is a fair exchange scheme in which exchanges between peers that are at most distance two hops apart.

Our first weak condition, the *neighborhood sufficiency condition*, is that the sum of the data in the neighborhood of each peer be at least equal to the size of the data that peer needs to backup. The second condition, the *even parity condition*, is that each peer have an even number of units. In reality, this is not expected to occur naturally; however, any peer with an odd number of units could easily generate an “empty” unit to satisfy this condition and give a very close approximation to fairness.

Notice that in Figure 1.2 where the $f$-sufficiency condition holds but the even parity condition does not, a fair exchange scheme does not exist, even when allowing two hops. The pairs (1, 2), (3, 4), and (5, 6) all exchange a single unit, which leaves 3 units ($f(7)$, $f(8)$, and $f(n)$) left to be exchanged. Clearly, no fair exchange scheme exists in this case.

Let the $f$-value of each vertex in Figure 1.2 be 2. The pairs (1, 2), (3, 4), and (5, 6) all exchange 2 units, which leaves 6 units ($f(7)$, $f(8)$, and $f(n)$) left to be exchanged. Now, the pairs (7, 8), (7, $n$), and (8, $n$) exchange a single unit to complete the fair two-hop exchange scheme.
1.2.1 Graph Theoretical Model

It is easily verified that the fair exchange scheme shown in Figure 1.1 corresponds to an integer $f$-factor as defined in Chapter 2. In this thesis, we introduce the definition of two-hop $f$-factors, which correspond to our FTHES. Let a two-hop $f$-factor in $G$ be an integer $f$-factor in $G^2$. The square of $G$ is the graph $G^2 = (V(G), E_k)$ where $pq \in E_k$ if and only if the hop-distance from $p$ to $q$ is less than or equal to 2. Further discussion of $f$-factors is left to Chapter 2.

1.2.2 Theorems and Results

In Chapter 3, we show that, through a simple transformation, the FTHES can be computed using existing $f$-factor theory and algorithms. The main result in Chapter 3 is our fundamental existence theorem, which shows that an FTHES always exists when the neighborhood sufficiency and even parity conditions hold. Lastly, we give an approximation to fairness when the even parity condition is relaxed.

In Chapter 4, we present an algorithm for computing an FTHES when the neighborhood sufficiency and even parity conditions hold. We then show that this algorithm has a linear time complexity of $O(n + m)$, in terms of the number of vertices and edges, $n$ and $m$, respectively. In Chapter 5, we present a distributed version of this algorithm and show its communication (message) complexity.
In Chapter 6, we discuss the size (number of exchanges) of an FTHES. We prove that the total number of exchanges in an FTHES is no more than $2n - 3$. This shows that, on average, peers are involved in at most four (4) exchanges. We also present simulation results which show that the total number of exchanges in practice is expected to be less than $2n - 3$. As part of our future work, we conjecture that this could be reduced to $n$. Last, we show that finding an FTHES of size $k$ is NP-hard via a reduction to the sum of subsets problem.

In Chapter 7, we present an application called MyBook, which motivated the FTHES work. MyBook allows users to compile, organize, and structure topic-focused content through a graphical user interface and a novel data structure called MBK. In a proposed P2P model, MyBook could be used to support the formation of groups of peers interested in common topics and allow them to collaborate on those topics by sharing relevant topic-focused content.

Finally, we conclude and provide several different future research directions. This includes a conjecture on the minimum size of an FTHES and a variant of the FTHES problem which capacitates the amount of data exchanged between peers.
Chapter 2

Background and Related Research

In this chapter, we discuss relevant background information and related research pertaining to this work. The research that is discussed in the following chapters subsumes several different areas of research including peer-to-peer computing and graph theory - $f$-factors in particular. Each of these topics is touched upon in the following sections. Additionally, at the end of this Chapter, we provide definitions for several terms and concepts that are used in later Chapters.
2.1 P2P

P2P networks have been applied to myriad areas of research including file sharing, distributed computing, backup, archiving, anonymous web browsing, and media streaming to name a few. P2P networks are an attractive alternative to the more traditional server-client model. In the traditional server-client model, nodes on the Internet act as either a server or a client but not both. P2P networks allow nodes to act as both a server and client - a *servant*. This eliminates (or at least reduces) the need for expensive dedicated server components. It also eliminates these dedicated servers as single points of failure as their duties are spread to each node.

P2P networks can be either structured or unstructured. Structured P2P networks are typically thought of as overlays. While the underlying network has no structure, an overlay in the form a ring or mesh can be “applied” to induce structure. This allows data to be distributed and obtained via protocols which guarantee a deterministic number of steps to put or get data. Examples of structured P2P networks include Chord [38], CAN [33], and Pastry [36].

Unstructured P2P networks are random in the sense that peers choose “neighbors” arbitrarily. The protocols for retrieving data are not deterministic. Instead, they rely on broadcasting to flood the network with requests for data. It is possible that the requested data will not be found in the case that the request does not reach the node(s) that have that data before a cer-
tain timeout or hop-distance. Without timeout or hop-distance restrictions, requests could traverse the network indefinitely without success and choke its performance. The most notable unstructured P2P network is Gnutella [14].

2.1.1 P2P File Sharing

Arguably the most popular application of P2P networks is file sharing. Napster [29] burst onto the scene in 1999 as a new way to share music and led to myriad other file sharing networks including BitTorrent (uTorrent, Transmission) [8, 32], Gnutella (Limewire, BearShare) [5, 14], Freenet [7], FastTrack (Kazaa [19]).

2.1.2 Backup/Archiving

We now discuss three P2P backup systems. Each has characteristics that are desirable in our model.

pStore

pStore [3] is a secure P2P backup system proposed by several students at MIT. It makes use of the unused space on its users hard disks to provide reliable, secure, and efficient backups. Reliability is provided through a replication scheme where data is split into chunks, and the chunks are duplicated and stored in multiple locations. These chunks are secured via hashing and
symmetric key encryption. The encryption not only allows the owner to verify that its data is intact but also serves as a mechanism that allows the owner to delete its data. pStore uses Chord as its P2P storage protocol.

The scheme presented in this work is quite similar to pStore in that it relies on users having unused disk space to fuel the backups. Security is also desirable via hashing and encryption; however, it is not a major focus in this context. Another common feature is the concept of splitting data into blocks. Our scheme differs from pStore primarily in how the blocks are distributed.

**Pastiche**

Pastiche [10] is a P2P backup solution based on Pastry, data-based indexing, and convergent encryption. Pastry is a scalable, self-organizing, P2P routing and object location protocol. data-based indexing allows Pastiche to identify common content such that multiple copies of that data are not backed up. Convergent encryption facilitates the content-based indexing by allowing each peer to use the same encrypted file(s) without having to share keys. Like pStore, Pastiche splits files into chunks and relies on content-based indexing to do so. A similar technique could be applied to this research; however, it is not a focus.

Pastiche introduces the concept of “backup buddies.” Each peer identifies a set of backup buddies with whom it shares data. It is suggested that each peer maintain five buddies. Finding backup buddies is a combination of identifying other peers with similar data (overlap) to reduce storage needs
and geographical diversity. This is quite similar to our scheme in that the buddies exchange data for backup; however, exchanges (backup buddies) are governed by hop-distance in our scheme.

**PeerStore**

PeerStore [22] is a P2P backup solution based on both DHTs and unstructured networks. Each peer splits its files into data blocks which are encrypted and have unique identifiers. A file is represented by a list of unique data block identifiers.

The key idea of PeerStore is the decoupling of metadata management from data block storage. Here, metadata refers to the tracking information for each data block, and this information is managed using DHTs. The use of DHTs for metadata management is advantageous in that it supports efficient search and allows for duplicate detection. An unstructured network is used for the actual distribution of the data blocks.

PeerStore is similar to our model in that files are split into blocks. Additionally, it uses symmetric block trading where peers search for partners to trade blocks with. A key difference from our model is that PeerStore does not require equal trades. Instead, it tracks a *trade ratio* which allows peers to accept imbalanced exchanges.
2.1.3 Trust and Reputation

An important feature of any P2P storage solution is trust. In many situations, it is vital that peers not be able to read, modify, or delete any data that they are storing for other peers. Any number of encryption techniques, such as the Advanced Encryption Standard (AES) [39], could be used to protect the confidentiality and integrity of the data.

Reputation may be used to determine the trust that one peer has for another. Each peer carries a reputation value measured in any number of ways. Peers shown to have behaved badly (e.g., deleted or otherwise modified data stored for other peers) in the past have a diminished reputation and are therefore less likely to be trusted. Several reputation schemes could be used in our P2P model including Eigentrust ([16]).

2.1.4 Fairness

Fairness is a widely-studied topic in P2P networks. Much of the research revolves around the issue of free-riding - the case where peers consume but do not contribute resources. This is a rather common problem in P2P file sharing applications and sparked the need to introduce incentives. For example, Samsara [9] suggests a protocol where peers contribute storage equivalent to the amount they store.

Fairness is also associated with load balancing. A distribution of data in a P2P system that overloads one or more peers may be considered unfair.
This is an especially important feature of our model.

Many P2P systems use distributed hashing (distributed hash table (DHT)) to distribute data. The DHT data structure is based on consistent hashing [17] and benefits from its favorable load balancing properties. The use of consistent hashing in DHTs balances the storage load amongst peers with at most $O(\log n)$ imbalance, where $n$ is the number of peers in the system.

Refinements to consistent hashing have been made in recent research to improve its load balancing properties. In practice, the application of consistent hashing may not split the DHTs address space evenly, making it likely that one or more peers will be responsible for a larger portion of it. Even when the address space is split evenly, some applications may not randomize the placement of items into the space. In both cases, imbalance can occur. The authors in [18] address both of these issues.

Another approach to load balancing in P2P systems is the randomized choose-two algorithm developed in [6, 26]. In this algorithm, two servers are chosen at random. The server with the lowest load, in terms of storage or processor usage for example, is assigned the item. They prove that choosing $n > 1$ servers is exponentially better than choosing only one server and that this behavior is apparent for small $n$ (i.e., 2).

### 2.1.5 Replication and Redundancy

Replication in P2P systems is typically accomplished by using one of the following methods:
1. Application of \( n \) distinct hash functions to put an item in \( n \) distinct locations when using DHTs.

2. Erasure coding to introduce redundancy.

Systems such as PAST and Chord use (or suggest using) the first method for data replication. In PAST [37], when files are inserted into the system, they are replicated \( k \) times. The authors in [36] describe a system built on PAST and perform location and routing analysis where \( k = 5 \). In the description of the Chord protocol in [38], Stoica et. al. state that data could be replicated by storing it under multiple distinct Chord keys.

OceanStore uses erasure coding, such as interleaved Reed-Solomon codes [31] or Tornado codes [23], to introduce redundancy into their P2P system [21]. The fragments produced by this technique are stored throughout the system to support high availability. Data can then be retrieved by requesting more fragments than actually needed and reconstructing it as soon as enough fragments have arrived.

Our storage model could make use of any one of these replication/redundancy techniques, but is better suited to the models that split data into unit-sized blocks. More information regarding replication/redundancy in P2P systems can be found in [23, 21, 37, 31, 34, 35, 36, 38].
2.2 Graph Theory

2.2.1 f-Factors (Integer vs 0/1)

The notion of f-factors is the basis for many of the theorems and algorithms discussed in the Chapters that follow.

Let $G = (V, E)$ be an undirected graph with a vertex set $V$ and an edge set $E$. Let $f : V \to Z^*$ be a mapping of the vertices to a set of non-negative integers [42], where $Z^*$ is the set of all non-negative integers. An integer $f$-factor of $G$ is a mapping $\phi : E(G) \to Z^*$ such that $\sum_{pq \in E} \phi(p, q) = f(p)$. Figure 2.1 shows an example of an integer $f$-factor.

The case where for each edge $e \in E$, $\phi(e) \leq 1$, is a more traditional definition of the $f$-factor, or a 0/1 $f$-factor. Notice that the integer $f$-factor problem can be solved as a 0/1 $f$-factor when each edge $e \in E$ is allowed to be a multi-edge; however, this may affect the complexity of the algorithms.

Throughout this thesis, we use the term $f$-factor to mean integer $f$-factor; unless otherwise stated. Note that $\phi(p, q)$ corresponds to the size of the exchange between $p$ and $q$ in a 1-hop exchange scheme.

Much work has been done to prove the existence of $f$-factors of graphs. Tutte showed the necessary and sufficient conditions for the existence of 1-factors, which are more commonly called perfect matchings.

**Theorem 2.1** (Tutte's 1-Factor Theorem). A graph $G$ has a 1-factor if and only if $\forall S \subset V(G)$, $C_{odd}(G - S) \leq |S|$, where $C_{odd}$ is the number of components with an odd number of vertices.
Through a transformation, Tutte's 1-Factor condition can be applied to obtain his famous $f$-factor theorem.

A 1-factor corresponds to a perfect matching. More generally, an $f$-factor corresponds to a perfect $b$-matching where $b = f$. Therefore, existing $b$-matching theory and algorithms are relevant and can be used to solve the $f$-factor problem. For convenience, instead of saying perfect $b$-matching, we use the term $f$-factor.

$b$-Matching Algorithms

Several algorithms for computing $b$-matchings exist. Most are based on a variant of primal-dual blossom algorithm, which uses a linear programming. Such algorithms start with an imperfect solution and augment it.
until a perfect $b$-matching is found. The best known $b$-matching algorithm is $O((n \log n)(m + n \log n))$, which is (strongly) polynomial ([1], [28]).

### 2.2.2 $k$-Hop $f$-Factors

In this thesis, we introduce the definition of $k$-hop $f$-factors. The $k^{th}$ power of a graph $G$ is the graph $G^k = (V(G), E_k)$ where $pq \in E_k$ if and only if the hop-distance from $p$ to $q$ is less than or equal to $k$ [41]. A $k$-hop $f$-factor in $G$ is an $f$-factor in $G^k$.

Here, $k$ corresponds to the maximum hop-distance of an exchange in our model. In this thesis, we restrict $k = 2$, which is the square of a graph, $G^2$.

### 2.3 Terminology

The following definitions will be used throughout this dissertation.

**Definition 2.1.** Let $f(p)$, or the $f$-value of $p$, be the size of the data (measured in units) that $p$ has to backup.

**Definition 2.2.** Let $\eta(p) = \sum_{pq \in E(G)} f(q)$ be the neighborhood $f$-sufficiency, or simply neighborhood sufficiency, of $p$.

**Definition 2.3.** We define a non-strict local maxima as any peer $p$ for which $f(p) \geq f(q), \forall pq \in E(G)$.

**Definition 2.4.** If $f(p) > f(q), \forall pq \in E$, we call $p$ a strict local maxima.

**Definition 2.5.** When $f(p) = 0$, we say that $p$ is satisfied.
Chapter 3

Fair Two-Hop Exchange

Scheme (FTHES)

In this Chapter, we show that an FTHES can computed in (strongly) polynomial time, or shown not to exist, by reducing the problem to finding an $f$-factor in the square of $G$. The main result of this Chapter is an existence theorem which shows that there always exists an FTHES provided that the neighborhood $f$-sufficiency and even parity conditions hold. As we will see in a subsequent Chapter, this existence theorem will lead to an efficient sequential algorithm and a practical distributed algorithm.
3.1 Notation and Definitions

We start by introducing some notation and several definitions. Recall that pairs of peers exchange blocks in our exchange scheme. We define an exchange as follows:

**Definition 3.1.** An exchange, $\varepsilon(p,q)$, is the size of the block (units of data) that a peer $p$ sends to another peer $q$ for storage. If $\varepsilon(p,q) = \varepsilon(q,p)$, we say the exchange is *fair*.

Clearly, an important goal of our exchange scheme is to backup all of the data in the P2P network. A set of fair exchanges such that all of the data is backed up is called an *$f$-exchange scheme* and is defined as follows:

**Definition 3.2.** An $f$-exchange scheme is a set of exchanges such that $\forall p \in V$, $\sum_{pq \in E} \varepsilon(p,q) = f(p)$. If all exchanges are fair, then we say that the $f$-exchange scheme is *fair*.

As we pointed out earlier, an efficient way to exchange data is to do so between pairs of peers that are close. This has several benefits including increasing trust and improving network efficiency. Exchanging between adjacent peers (friends) may be efficient; however, this restriction makes it likely that we will be very limited in the amount of data that can be exchanged (see Figure 1.2). Therefore, we consider exchanges between peers that are at most 2-hops, or between a friend of a friend. We call this a *fair two-hop exchange scheme*, or FTHES.
**Definition 3.3.** A $k$-hop exchange, $\varepsilon(p, q)$, is an exchange between $p$ and $q$ that are at most $k$-hops apart.

**Definition 3.4.** A $k$-hop $f$-exchange scheme is a set of exchanges, $F$, between peers that are at most $k$-hops apart, i.e., $\varepsilon(p, q) = 0$ when $\text{dist}(p, q) > k$. Again, if all exchanges are fair, we say *fair $k$-hop $f$-exchange scheme*.

### 3.2 Modeling Fair Two-Hop Exchange Scheme with Two-Hop $f$-factors

Let $G = (V, E)$ be a graph with a vertex set $V$ and an edge set $E$. Recall from Chapter 2, that a $k$-hop $f$-factor in $G$ is an $f$-factor in the $k^{\text{th}}$ power of $G$, i.e., $G^k$. This corresponds to allowing at most $k$-hops between vertices. Note that an FTHES in $G$ corresponds to a two-hop $f$-factor in $G$.

**Proposition 3.1.** The problem of finding a two-hop $f$-factor is computationally equivalent to that of finding an $f$-factor.

**Proof of Proposition 3.1.**

To prove this theorem, we will show that the problem of finding an $f$-factor can be reduced to the problem of finding a two-hop $f$-factor, and vice-versa.

$f$-factor problem $\longrightarrow$ two-hop $f$-factor problem

The $f$-factor problem can be reduced to the two-hop $f$-factor problem by inserting a bivalent vertex with $f = 0$ between each vertex of $G$. In this case,
we are forced to use two-hop paths to find an $f$-factor - a two-hop $f$-factor. See Figure 3.1.

\textit{two-hop $f$-factor problem} $\rightarrow$ \textit{$f$-factor problem}

The two-hop $f$-factor problem can be reduced to the $f$-factor problem by squaring the graph $G$ and finding an $f$-factor in it. Notice that in Figure 3.1, the square of $G'$ is congruent to the original $G$ if we do not consider the 0 weighted vertices and their edges.

Through a transformation of the FTHES problem to the two-hop $f$-factor problem, we have shown that existing $f$-factor algorithms can be used to compute an FTHES or show that one does not exist. While efficient algorithms are known for computing $f$-factors (i.e., perfect $b$-matchings), they tend to be sophisticated and difficult to implement. As such, there are a limited
number of implementations ([28]), and the implementations that do exist do not lead to practical distributed algorithms.

In the next section, we show that if our two weak conditions hold, an FTHES always exists. In later Chapters, we present a linear time algorithm and a practical distributed algorithm for computing an FTHES.

### 3.3 Fundamental Existence Theorem

Surprisingly, if we allow two hops and the neighborhood f-sufficiency and even parity conditions hold, an FTHES always exists. Given an FTHES, we define its set of exchanges as $F$, sometimes called the support of an FTHES.

**Theorem 3.1.** If the neighborhood $f$-sufficiency and even parity conditions hold, then there exists an FTHES in $G$.

**Proof of Theorem 3.1.** Let $G = (V, E)$ be an undirected graph with a vertex set $V$ and an edge set $E$. Let $f : V \rightarrow \mathbb{Z}^*$ be a function that maps each vertex $p \in V$ to an even non-negative integer. For all $p \in V$, $f(p) \leq \eta(p)$.

We give a constructive proof, which we present in two parts. Providing a constructive proof will facilitate the analysis of the size of the FTHES in a later Chapter and will more clearly lead to the efficient sequential and distributed algorithms that follow.

In the first part, we consider the strict local maxima in $G$ and compute exchanges between them, if necessary, so that no strict local maxima is within two hops of another. Then, we transform $G$ into an ascending forest $A$ and
compute exchanges in it using two different methods: one for vertices with $\text{depth}(p) > 2$ and the other for vertices with $\text{depth}(p) \leq 2$.

We start with $G$ as shown in Figure 3.2.

Unless otherwise noted, an exchange $\varepsilon(p, q)$ results in $f'(p) = f(p) - \varepsilon(p, q)$ and $f'(q) = f(q) - \varepsilon(p, q)$.

**Part 1 - Exchanges involving Strict Local Maximas**

First, we consider the strict local maxima in $G$ and add exchanges to $F$ so that no strict local maxima is within two hops of another. Let $S$ be the set of strict local maxima in $G$. Consider two strict local maxima $s_1$ and $s_2$ and a vertex $p$ such that $s_1, p$ and $s_2, p \in E(G)$. For each such pair of strict local maxima within two hops of another, we will show that one or both are reduced to non-strict local maxima.

Let $\text{max}_f(s)$ be a vertex having (potentially sharing) the maximum $f$-
Figure 3.3: Strict local maxima (marked in double circles) within two hops of one another. Making the exchange $\varepsilon(1, 8) = 2$ reduces vertex 8 to a non-strict local maxima.
value in the neighborhood of $s$. Notice that after an exchange of size $f(s) - f(max_f(s))$, a strict local maxima $s$ becomes non-strict. For convenience, let $\rho(s) = f(s) - f(max_f(s))$. Also, notice that an exchange larger than $\rho(s)$ potentially compromises the neighborhood $f$-sufficiency condition for $max_f(s)$.

Let $s_1$ and $s_2$ be two strict local maximas that are two hops apart. Clearly, the maximum size of the exchange $\varepsilon(s_1, s_2)$ that guarantees the preservation of the neighborhood $f$-sufficiency condition in $G$ is $min\{\rho(s_1), \rho(s_2)\}$. When $\varepsilon(s_1, s_2) = \rho(s_1)$, the exchange results in $s_1$ becoming a non-strict local maxima. When $\varepsilon(s_1, s_2) = \rho(s_2)$, $s_2$ becomes a non-strict local maxima. In the case that $\rho(s_1) = \rho(s_2)$, $\varepsilon(s_1, s_2)$ results in both $s_1$ and $s_2$ becoming non-strict local maximas.

Applying this to each pair of strict local maximas that are within two hops of one another eliminates all such occurrences. In our $G$ shown in Figure 3.3, vertices 1 and 8 are strict local maximas. Here $\varepsilon(1, 8) = min\{4, 2\} = 2$ reduces vertex 8 to a non-strict local maxima.

**Part 2 - Remaining Exchanges via Ascending Forest**

Before continuing, we introduce a strict total ordering over the vertices of $G$. Each vertex is assigned a unique integer $ID$. For simplicity, we assume this to be $(1, \ldots, n)$, where $n$ is the number of vertices. Let the strict total order over $v \in V(G)$ be defined as follows:

- $p > q$ iff $f(p) > f(q) \lor f(p) = f(q) \land ID(p) > ID(q)$

Given the previous exchanges, we can assume that no strict local maxima
is within two hops of another. Now, we will compute the remaining exchanges using an ascending forest. Here we use the term *ascending* to describe the fact that $p$ is a parent of $q$ if $f(p) \geq f(q)$. Let $A$ be the ascending forest built from $G$ by allowing each $p \in V(G)$ (considered in the order $1, 2, \ldots, n$) choose a parent, $\text{parent}(p)$, from the vertices in its 1-neighborhood using the following rules (in order of preference):

1. A strict local maxima (there could be a maximum of 1 per the first exchange step)

2. $q = \text{max}_f(p)$ iff $\text{parent}(q) \neq p$

3. null (i.e., $p$ becomes a root)

It is possible that a strict local maxima in the neighborhood of $p$ is not $\text{max}_f(p)$ so we force that selection. The next preference is to choose $\text{parent}(p) = \text{max}_f(p)$; however, to prevent cycles, we must take care that $\text{parent}(\text{max}_f(p)) \neq p$ (i.e., $p$ is not the parent of $\text{max}_f(p)$ already). In the case that either of the previous preferences can not be satisfied, $p$ becomes a root in $A$.

Notice that the $f$-sufficiency condition holds in $A$. In fact, the $f$-sufficiency condition is satisfied by the parent alone of each vertex below the root. This means that the $f$-sufficiency condition need only be satisfied for strict local maxima.

With our ascending forest $A$ built (shown in Figure 3.4), we will use two different ways to compute the remaining exchanges. We first compute
Figure 3.4: Ascending forest $A$ built from $G$ and $f'$. 
Figure 3.5: The vertices in $A$ having depth at least 2.

exchanges between vertices $p \in V(A)$ for which $\text{depth}(p) \geq 2$. Then we finish by computing exchanges between vertices $p \in V(A)$ for which $\text{depth}(p) \leq 2$.

First, we will satisfy the vertices in $A$ having $\text{depth}(p) > 2$ (shown in Figure 3.5). Let $p$ be a vertex in $A$ with $\text{depth}(p) \geq 2$, and let the children of $p$, $q \in Q$, all be leaf vertices. We add exchanges to $F$ as follows to satisfy each $q \in Q$ (and possibly $p$). This sub-part can be applied to each $p$ with $\text{depth}(p) \geq 2$ until the maximum depth in all of $A$ is at most 2.
For convenience, we sort \( q \in Q \) using the strict total order defined above. If \(|Q| = 1\), we add the exchange \( \varepsilon(p, q_1) = f(q_1) \) to \( F \). Then, \( f'(q_1) = 0 \) and \( f'(p) = f(p) - f(q_1) \). Now, \( q_1 \) has been satisfied and is removed from \( A \) to form \( A' \). When \( f'(p) = 0 \), \( p \) has been satisfied as well and is also removed from \( A \).

If \(|Q| \geq 2\), add the exchange \( \varepsilon(q_i, q_{i+1}) = f(q_i) \) to \( F \). \( f'(q_i) = 0 \) and \( f'(q_{i+1}) = f(q_{i+1}) - f(q_i) \). Again, since \( q_i \) is satisfied, it is removed from \( A \). If \( q_{i+1} \) is satisfied as well (\( f'(q_{i+1}) = 0 \)), it is also removed from \( A \) to form \( A' \). This action is repeated until either \(|Q| = 0\) or \(|Q| = 1\), which we discussed above.

After this sub-part, each vertex \( p \) with \( \text{depth}(p) > 2 \) is satisfied, and \( A' \) is made up of ascending trees having a depth at most 2.

In Figure 3.5, the following exchanges, in order, satisfy each vertex with depth greater than 2: \( \varepsilon(9, 3) = 2 \), \( \varepsilon(3, 6) = 2 \), \( \varepsilon(6, 8) = 2 \), and \( \varepsilon(8, 5) = 4 \). After this step, we are left with \( A \) shown in Figure 3.6.
Finally, we satisfy the remaining vertices in $A$ (having $\text{depth}(p) \leq 2$). Before discussing this part, we introduce a few notations. Let $r$ be the root of an ascending tree $t$, $t \in A$. Let the children of $r$ be denoted by $P$ and the grandchildren of $r$ denoted by $Q$ as shown in Figure 3.6. Finally, let $\eta_2(r)$ be the $f$-sufficiency in the 2-neighborhood of $r$.

For each ascending tree $t \in A$, we add exchanges to $F$ as follows. Notice that if $f(r) = \eta_2(r)$, we augment $F$ by adding $\varepsilon(q_i, r) = f(q_i)$ for each $q_i \in Q$ and $\varepsilon(p_i, r) = f(p_i)$ for each $p_i \in P$. This satisfies $r$ and each of the vertices in $P$ and $Q$. Additionally, this results in $A' = A - t$ (i.e., the ascending tree $t$ has been satisfied).

Now we consider the only other case where $f(r) < \eta_2(r)$. Recall that if $f(r) > \eta_2(r)$, the neighborhood capacity restriction must not have applied to the original graph $G$.

First, notice that an exchange $\varepsilon(p, q) = f$ reduces $\eta_2(r)$ by $2f$. Also notice that $\eta_2(r) - f(r)$ is even. With these two facts, we can add exchanges to $F$ until $\eta_2(r) = f(r)$, which we solved above. Starting with the vertices in $W$, add exchanges as described in the previous exchange step, with the caveat that the size of each exchange be $\varepsilon(p, q) = \min\{f(p), (\eta_2(r) - f(r))/2\}$.

For each vertex $p$ where $\varepsilon(p, q) = f(p)$, $p$ is completely satisfied and is removed from $t$ ($t' = t - p$). Then, $\eta_2(r)$ is reduced by $2f(p)$. When $\varepsilon(p, q) = (\eta_2(r) - f(r))/2$, the exchange results in $\eta_2(r) = f(r)$, which was solved above.

The following exchanges satisfy the ascending forest shown in Figure 3.6.
We start with $\eta_2(1) = 16$.

1. $\varepsilon(5, 2) = 2; \eta_2(1) = 12, 12 > 10$: continue

2. $\varepsilon(4, 7) = 1; \eta_2(1) = 10, 10 = 10$: all remaining vertices exchange with vertex 1

The exchanges from the first part that eliminated strict local maximas within two hops of another together with the exchanges in the ascending forest $A$ form the support of the FTHES, $F$, that satisfies each $p \in V$. This concludes our proof of Theorem 3.1.

**Claim 3.1.** There exists a sequential algorithm that can compute an FTHES in at most $O(n + m)$ steps.

*Proof of Theorem 3.1.* The proof of this claim is shown in the next Chapter.

---

### 3.4 Near-Fair Exchange Scheme

If we relax the even parity condition, we can compute a *near-fair* two-hop exchange scheme (NFTHES) that approximates fairness. Let $k$ be the disparity of an exchange - $k = |\varepsilon(p, q) - \varepsilon(q, p)|$. We use the term near-fair exchange to describe this case.

We will now show that when the neighborhood sufficiency condition holds and the even parity condition is relaxed, $k = 1$. To do so, we will describe
a slight modification of our constructive proof for our FTHES. Additionally, we will prove that the maximum number of near-fair exchanges when \( k = 1 \) is \( \frac{n}{3} \).

The basis for our NFTHES is a slight modification of the constructive proof for the fundamental existence theorem. The exchanges between strict local maximas within two-hops of one another, the construction of the ascending forest, and the exchanges involving peers at depth greater than two (2) in the ascending forest are all the same as in the previous FTHES discussion. Clearly, our modification will involve the way that exchanges are computed between peers that are at depth less than or equal to two (2).

Let \( r \) be the root of an ascending tree in the ascending forest. In the FTHES constructive proof, peers at depths 1 and 2 exchange until \( \eta_2(r) = f(r) \). With the even parity condition relaxed, \( \eta_2(r) + f(r) \) can potentially be odd. In this case, peers at depths 1 and 2 exchange until \( f(r) = \eta_2(r) - 1 \). At this point, the remaining peers exchange the rest of their \( f \) with \( r \). Exactly one of these exchanges is near-fair. This single near-fair exchange leaves \( r \) with an imbalance of 1.

Now that we have shown that each ascending tree in the ascending forest can have at most one near-fair exchange, we must show that the maximum number of near-fair exchanges is \( \frac{n}{3} \). Notice that the minimum size of an ascending tree that has a near-fair exchange is 3. If it was 2, the neighborhood sufficiency condition would not hold. The maximum number of ascending trees constructed from our original graph is \( \frac{n}{3} \). In this case, each ascending
tree in the ascending forest contains exactly 3 peers.
Chapter 4

Algorithm for Computing an FTHES

We now describe our algorithm for finding an FTHES. The algorithm extends the constructive proof of our existence theorem given in the previous Chapter. We show the practicality of implementing our algorithm in practice by providing psuedo-code. Finally, we provide a worst-case complexity analysis of the algorithm, which is shown to be $O(n + m)$. 
4.1 Outline

<table>
<thead>
<tr>
<th>Data: Graph $G = (V, E)$, $f : V(G) \rightarrow \mathbb{Z}^+$, $f(p)$ even, and $f(p) \leq \eta(p)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result: $F$ (FTHES support)</td>
</tr>
<tr>
<td>1 Compute exchanges so that no strict local maxima is within two hops of another;</td>
</tr>
<tr>
<td>2 Build ascending forest $A \subset G$;</td>
</tr>
<tr>
<td>3 Compute exchanges in each ascending tree so that each peer is satisfied;</td>
</tr>
</tbody>
</table>

Algorithm 1: Algorithm to Compute a FTHES

4.2 Auxiliary Functions

MaxFNeighborOf(Graph $G$, Vertex $p$) This function returns the vertex $q$ in the neighborhood of $p$ that has the largest $f$-value such that $f(q) > f(p)$; otherwise, it returns $null$. If more than one vertex has the largest $f$-value, we choose the vertex with the largest UID to break the tie.

MaxFReductionOf(Graph $G$, Vertex $p$) This function returns the difference between $f(p)$ and $f$(MaxFNeighborOf($G$,p)).

ComputeFInTwoNeighborhoodOf(Graph $G$, Vertex $p$) This function returns the sum of the $f$-values in the two-neighborhood of $p$ in $G$. 

44
4.3 Exchanges Involving Strict Local Maximas

The first step in our algorithm is to identify exchanges between strict local maxima such that no strict local maxima is within two hops of another. Note that strict local maxima cannot be adjacent or they would be non-strict, per our definition of the term. Identifying these exchanges is done in two parts. First, we find the set of strict local maxima. Then, we compute the exchanges between those strict local maxima that are within two hops of one another.

4.3.1 Identify Strict Local Maximas

We begin by identifying the strict local maxima in $V(G)$ to form the set $S$. For each vertex $p \in V$, we inspect each adjacent vertex $q$ and compare their $f$-values. If $f(p) > f(q)$, for all $q$, $pq \in E$, $p$ is added to $S$. This procedure is very similar to the breadth-first search procedure (BFS).

4.3.2 Computing Exchanges

With the strict local maxima identified, we now show a procedure that computes exchanges, if necessary, such that no strict local maxima is with two hops of another. First, we identify all vertices that have at least two strict local maxima in their neighborhood to form the set $I$. Clearly, if
$|I| < 1$, no strict local maxima is within two hops of another.

Now, we will process each $i \in I$ and compute exchanges so that at most a single strict local maxima remains connected to each $i$. Recall from the discussion in Chapter 3 that no strict local maxima can be reduced to anything less than a non-strict local maxima. For this reason, we will compute the amount that each strict local maxima can exchange and still be at least a non-strict local maxima. This value is computed via $MaxFReductionOf$ and stored in the $maxReduction$ property of each strict local maxima.

We use the following procedure to “eliminate” each $i \in I$. 
Input: Graph $G = (V, E)$, $i$ a vertex connected to at least two strict local maximas, $S$ the set of strict local maximas

Output: $F$ - FTHES support (updated)

1. $\text{Vertex curS} = \emptyset$; /* current strict local maxima */
2. foreach $pq \in E(G)$ do
3.     if $p \in S$ then
4.         if $\text{curS is} \emptyset$ then
5.             $\text{curS} = p$;
6.         end
7.     else /* max exchange size for curS and p */
8.         int $\text{sizeOfExchange} = \text{Math.Min(curS.maxReduction, p.maxReduction)}$;
9.         $\text{F.Add(new Exchange(curS, q, sizeOfExchange))}$;
10.        $\text{curS.f'} = \text{sizeOfExchange}$;
11.        $\text{curS.maxReduction} - = \text{sizeOfExchange}$;
12.        $\text{p.f'} = \text{sizeOfExchange}$;
13.        $\text{p.maxReduction} - = \text{sizeOfExchange}$;
14.     /* curS reduced to non-strict */
15.     if $\text{curS.maxReduction} == 0$ then
16.         $\text{S.Remove(curS)}$;
17.         $\text{curS} = \emptyset$;
18.     end /* p reduced to non-strict */
19.     if $\text{p.maxReduction} == 0$ then
20.         $\text{S.Remove(p)}$;
21.     end
22. else
23.     $\text{curS} = p$;
24. end
25. $\text{curS} = \emptyset$;
26. end
27. return $F$;

Algorithm 2: Procedure ComputeExchangesBetweenStrictLocalMaximasConnectedTo

In ComputeExchangesBetweenStrictLocalMaximasConnectedTo we consider each vertex adjacent to $i$. If we find an adjacent vertex $p$ that is a strict local maxima, we have two options. If $\text{curS}$ is not set, $p$ becomes
Otherwise, we compute an exchange that reduces one or both of $curS$ and $p$ to a non-strict local maxima. After considering each vertex adjacent to $i$, $i$ is adjacent to at most one strict local maxima.

### 4.4 Building an Ascending Forest

At this point, no strict local maxima is within two hops of another and our neighborhood sufficiency condition still holds. Our next step is to build an ascending forest that will facilitate the remaining exchanges in $G$. To build the ascending forest, we will choose $parent(p)$ for each $p \in V$.

Recall the rules for choosing $parent(p)$ that were discussed in Chapter 3. For each $p \in V$, the first preference is to choose any strict local maxima adjacent to $p$. Note that only one strict local maxima could exist in the neighborhood of $p$ per the exchanges computed above. If no such vertex exists, we choose $parent(p) = MaxFNeighborOf(p)$. Here, we must be careful to not create cycles so we make sure that the parent of $parent(p)$ is not $p$. Finally, if no such vertex exists, we choose $parent(p) = NULL$, which is equivalent to setting $p$ as a root in the ascending forest.

### 4.5 Computing Remaining Exchanges

Finally, we identify that remaining exchanges and add them to our FTHES support $F$. To do so, we will use two difference procedures which process
vertices having \( \text{depth} \geq 2 \) and \( \text{depth} \leq 2 \). The \textit{ComputeExchangesInAscendingTree} procedure is used to compute the exchanges in an individual ascending tree and is repeated to process the entire ascending forest. This could easily be parallelized since there is no interference between trees in the forest.

The \textit{ComputeExchangesInAscendingTree} calls two other procedures to process the “low” and “high” vertices. \textit{ComputeExchangesInAscendingTreeLow} processes vertices at or below depth 2, and \textit{algComputeExchangesInAscendingTreeHigh} processes vertices at or above depth 2.

### Algorithm 3: Procedure \textit{ComputeExchangesInAscendingTree}

**Input:** \( T \) - ascending tree, \( DD \) - the depth dictionary, \( F \) - FTHES support  
**Output:** \( F \) - updated FTHES support containing exchanges in \( T \)  

\[
\begin{align*}
1 & \quad F \leftarrow \text{ComputeExchangesInAscendingTreeLow}(T, DD, F); \\
2 & \quad F \leftarrow \text{ComputeExchangesInAscendingTreeHigh}(T, DD, F); \\
3 & \quad \text{return } F;
\end{align*}
\]

4.5.1 Exchanges Involving Vertices at Depth \( \geq 2 \)

The \textit{ComputeExchangesInAscendingTreeLow} procedure is used to compute exchanges between vertices at or below depth 2 in the ascending forest. In it, we use a postorder traversal. Clearly, in a postorder traversal, a vertex is not visited until all of its children have been visited. This allows us to process the ascending forest from the bottom up.

Our postorder traversal has a few important modifications. It computes
the depth of each vertex during a visit, with the root having a depth of 0. The
depth is used in this procedure as well as the next, which considers the top
3 levels of the ascending forest. The traversal does not compute exchanges
when visiting any vertex $p \in V$ for which

- $\text{depth}(p) < 2$
- $f(p) = 0$
- $\text{children}(p) = \emptyset$

Let $p$ be a vertex that is being visited for which $\text{depth}(p) \geq 2$, $f(p) > 0$, and $\text{children}(p) \neq \emptyset$. We use $C$ to denote the children of $p$. The following
procedure, VisitAndComputeExchangesFor, is executed for any such $p$ and
it computes the exchanges involving $C$ and possibly $p$. 


Input: \( p \) - a vertex for which \( \text{depth}(p) \geq 2, f(p) > 0 \), \( C \) - non-empty set containing the children of \( p \), \( F \) - FTHES support

Output: \( F \) - updated FTHES support containing exchanges involving \( C \) and possibly \( p \)

/* compute support for vertices at or below depth 2 */

1 curC ← \( \emptyset \);
2 foreach \( c \in C \) do
3     if curC = \( \emptyset \) then
4         curC ← \( c \);
5     end
6 else
7         sizeOfExchange ← Min(curC.Get(\( f' \)), c.Get(\( f' \))); // update f-values
8         curC.\( f' \) ← (curC.Get(\( f' \)) - sizeOfExchange);
9         c.\( f' \) ← (c.Get(\( f' \)) - sizeOfExchange);
10        // add exchange to \( F \)
11        \( F \).Add(new Exchange(curC, c, sizeOfExchange));
12        // curC has no more data to exchange
13        if curC.Get(\( f' \)) = 0 then
14            curC ← \( \emptyset \);
15        end
16        // c has more data to exchange
17        if curC = \( \emptyset \) and c.Get(\( f' \)) > 0 then
18            curC ← \( c \);
19        end
20     end
21 // last child has data to exchange with parent
22 if curC \( \neq \emptyset \) then
23     sizeOfExchange ← curC.Get(\( f' \));
24     \( F \).Add(new Exchange(curC, p, sizeOfExchange));
25     curC.\( f' \) ← (curC.Get(\( f' \)) - sizeOfExchange);
26     p.\( f' \) ← (p.Get(\( f' \)) - sizeOfExchange);
27 end
28 return \( F \);

Algorithm 4: Procedure VisitAndComputeExchangesFor
4.5.2 Exchanges Involving Vertices at Depth ≤ 2

**Input:** $T$ - ascending tree, $DD$ - the depth dictionary, $F$ - FTHES support

**Output:** $F$ - FTHES support for vertices at or above depth 2 in $T$

1. root ← $DD[0]$;

2. if root.Get($f'$) = $\eta_2$(root) then
   // exchange all remaining data with the root

3. // compute exchanges until root.Get($f'$) = $\eta_2$(root)

4. for $i$ ← 1 to 0 do

5.   foreach parent ∈ $DD[i]$ do

6.     // same exchange process as in
7.     // ComputeExchangesInAscendingTreeLow
8.     // with the exception of how the size of the
9.     // exchange is computed

10.    largestExchange ← Min(curC.Get($f'$), c.Get($f'$));
11.    sizeOfExchange ← Min(largestExchange,
12.        ($\eta_2$(root) − root.Get($f'$))/2);

13.    // add exchange to $F$ and update $f$-values and curC as
14.    // before

15.    if root.Get($f'$) = $\eta_2$(root) then
16.        // exchange all remaining data with the root
17.        break;
18.    end
19. end

20. end

21. return $F$;

**Algorithm 5:** Procedure ComputeExchangesInAscendingTreeHigh

52
4.6 Complexity Analysis

The worst case complexity of our algorithm is approximately $O(n + m)$. To show this, we will discuss the worst case complexity of each procedure used to compute our FTHES. Note that since this complexity is linear, we will omit the best and average complexity analyses.

4.6.1 SLMS

First, we identified the strict local maximas in $G$ using a procedure much like BFS, which required at most $n + 2m$ comparisons. Then, we identified all vertices ($i \in I$) that were connected to at least two strict local maximas. This required at most $n + 2m$ comparisons since we inspected each edge at most once from each end point.

Next, we computed the maximum block size that each strict local maxima could exchange before violating the neighborhood sufficiency condition. The worst case here is seen when there are $n - 2$ strict local maximas. In this case, computing this value would require $n - 2 + m$ comparisons because each strict local maxima would compute the value and all $m$ edges would have to be considered.

Then, we computed the exchanges between strict local maximas connected to each $i \in I$. The worst case for this operation is seen when $|I| = n - 3$. In this case, each edge is inspected exactly once resulting in $n - 3 + m$ comparisons.
4.6.2 Ascending Forest Operations

The first step in building the ascending forest was to have each vertex choose a parent. Here, each vertex compares itself to its adjacent vertices. Clearly, this requires exactly $n + 2m$ comparisons.

4.6.3 Exchanges in Ascending Forest

Computing the exchanges in the ascending forest was done using two procedures. Combined, these procedures looked at each edge in the tree at most twice. The postorder traversal used to compute exchanges for vertices at or below depth 2 required at most $2n - 1$ visits and the exchanges computed required at most $m$ operations. Processing the vertices at or above depth 2 required at most another $2n - 1 + m$. Therefore, the maximum number of comparisons is $4n - 2 + 2m$.

So, we have $(n + 2m) + (n + 2m) + (n - 2 + m) + (n - 3 + m) + (n + 2m) + (2n - 1 + m) + (2n - 1 + m) = 9n + 10m - 7$. Hence, we have a worst-case complexity of $O(n + m)$. 
Chapter 5

Distributed Algorithm for Computing an FTHES

We now describe our distributed algorithm for finding an FTHES. The algorithm extends the constructive proof of our existence theorem given in the Chapter 4 and uses many of the same functions described in the previous Chapter. Finally, we provide a worst-case analysis of the number of messages required to compute an FTHES in a distributed setting.

For convenience, we assume a synchronous network model. The synchronization could be supported by a number of different protocols including leader election [11]. Leader election involves electing a single peer as the “leader.” This leader would serve as a billboard where the other peers in the network could obtain and post information. For our purposes, the leader peer will not perform any computations other than those required to com-
pute exchanges involving itself or its adjacent peers. We will ignore further discussion of the complexities involved in this synchronous model in favor of focusing on the algorithm itself.

The algorithm is made of up a set of steps. Each step can have any number of rounds. A round is defined as an operation that all peers perform. Note that some peers may perform a “NOOP” or no operation if they have no work to do in a particular round.

5.1 Identify Strict Local Maximas

In the first step of our distributed algorithm, we identify any strict local maxima. First, each peer broadcasts its $f$-value to each of its neighbors in a single round. In the next round, each peer determines if it is a strict local maxima by comparing its $f$-value with that of its neighbors.

5.2 Remove Strict Local Maximas Within Two Hops

In the second step, the goal is to eliminate the case where any strict local maxima is within two hops of another. The communication involved in doing so is non-trivial and several rounds may be required. First, each peer broadcasts its strict local maxima status (1 if it is strict; otherwise 0) along with the minimum number of blocks required to reduce it to a local maxima.
\[(min \{ f(p) - f(q) \}, \forall pq \in E(G))\].

Next, any peer that receives more than one positive strict local maxima notification, broadcasts its status as an SLM helper to the strict local maxima in its neighborhood. These SLM helpers will facilitate the exchanges between strict local maxima. Clearly, if a strict local maxima peer receives such a broadcast, it is within two hops of another strict local maxima peer.

Now, any strict local maxima that has received an SLM helper notification responds to the SLM helper in its neighborhood with the largest UID. Any SLM helper that receives at least two such notifications, can then compute a set of exchanges that will reduce at least all but one of the strict local maxima peers in its neighborhood to non-strict local maxima. Finally, any remaining strict local maxima peers broadcast their strict local maxima status, and the process repeats until no SLM helper is found, which implies that no strict local maxima peer is within two hops of another.

In this step, the leader can be used by the peers not participating in the strict local maxima exchanges to monitor when the step is over. Any time a SLM helper is identified in a round, it posts its status to the leader, which keeps a count of the total number of SLM helpers. When the total number of SLM helpers is 0 at the end of a round, the next step can begin.
5.3 Ascending Forest Generation

In the next step, we build the ascending forest. Given that the $f$ values are likely to have changed after the exchanges involving strict local maxima peers, we use a round in which each peer broadcasts its $f$-value and strict local maxima status to its neighbors. Then, each peer chooses a parent per the selection rules specified in Chapter 3 by sending an parent request message to its selected parent peer.

Next, each peer sends a parent confirmation message to the peers that sent them parent request messages. There is a slight issue when considering two adjacent non-strict local maxima. Consider two such peers with UIDs A and B. Peer A chooses peer B, and peer B chooses peer A as its parent. If allowed, this would create a cycle. So, any peer that both sends and receives a parent request message to/from the same peer confirms the parent request only if the requesting peer’s UID is greater than its own. In our example, peer A would confirm peer B’s parent request so that B would become the parent of A.

5.4 Computing Exchanges

With the ascending forest generated, we now proceed to step where the remaining exchanges are computed such that we end up with an FTHES. The key to this step is to control the peers that compute exchanges in each round. Recall from the algorithm described in Chapter 4, that we have two
different methods for computing exchanges in the ascending forest - (i) peers at or below depth two and (ii) peers at or above depth two.

We begin by determining the depth of each peer in the ascending forest. Any peer not having a parent is a root of an ascending tree. These roots broadcast a depth message with a current depth of 0 to all of their children. These children then broadcast a depth message to all of their children with current depth +1. This continues until all peers know their depth.

The leader can be used in this step as a post for any depth messages sent. When a peer broadcasts its depth to its children, it also sends a notification to the leader. The leader does not track the current depth. Instead, it tracks the total number of depth messages sent in each round. Clearly, when that number is 0, every peer knows its depth.

Now that each peer knows its depth, we can proceed with the exchanges. The first step is to compute all of the exchanges between peers at or below depth 2 in the ascending forest. Here, any peer whose children are all either childless (leaf peers) or have already computed the exchanges for their children can compute exchanges for its children. To get this information, each peer sends its child count to its parent. Each such peer $p$ computes the exchanges for its children as described in Chapter 4. Then $p$ sends exchange instructions to each of its children and a completion message containing its updated $f$-value to its parent. This completion message allows the parent of $p$ to determine if it can compute exchanges for its children in the next round.

Once again, the peers post their activity to the leader peer. Specifically,
each peer computing exchanges in a round posts a message to the leader, which in turn keeps a count of the peers computing exchanges. When no peers are computing exchanges in this step, all of the exchanges between peers at or below depth 2 have completed their exchanges.

At this point, each peer at or above depth 2 in the ascending forest sends its $f$-value to its parent. Then, each peer sends the sum of its childrens’ $f$-values to its parent. Now, each root peer knows the sum of the $f$-values in its two-neighborhood.

Each root peer $r$ posts the largest UID of its children to the leader and sends that peer its current $\eta_2$. The other peers monitor this post and only those peers whose UID is posted compute exchanges. The active peers, which have knowledge of their parents’ $\eta_2$, compute exchanges as described in the algorithm from Chapter 4 taking care not to compute an exchange that forces $\eta_2(r) < f(r)$. Then, exchange instructions are sent to each child and an updated $\eta_2(r)$ is sent back to $r$. If $\eta_2(r) > f(r)$, $r$ posts the next largest UID of its children and another round is processed. When $\eta_2(r) = f(r)$, $r$ sends exchange instructions to each of its children to complete the exchanges for that ascending tree. A round in which the leader peer receives no UID signifies that the algorithm is complete.
5.5 Complexity

We will measure the complexity of this distributed algorithm in terms of the number of messages required to compute the FTHES. Note that this analysis is loose. The problem of finding a tighter bound is left as future research.

5.5.1 Messages Involving Strict Local Maximas

The first step was to compute exchanges between strict local maximas within two hops of one another. First, each peer broadcast its UID and $f$-value to its neighbors. This required $2m$ messages since a message was sent from both endpoints of every edge.

The second step consisted of a set of rounds in which the strict local maximas within two hops of one another were reduced to non-strict local maximas. In each round, the strict local maxima notification broadcast required at most $2m$ messages. The SLM helper notification step required at most $m$ messages - $m/2$ for the SLM helper messages and $m/2$ for the strict local maximas to select a SLM helper. Then, at most another $m/2$ messages to communicate the exchanges between the strict local maximas.

We know that at least one strict local maxima is reduced in each round, which gives $n - 3$ as the upper bound on the number of rounds. Each round required at most $\frac{3}{2}m$ messages. In total, $(n - 3) \times \frac{3}{2} = \frac{3}{2}nm - 3m$ messages were required to eliminate any case where a strict local maxima was within two hops of another.
5.5.2 Messages Involving Ascending Forest

Before selecting a parent, each peer sent its current $f$-value, which required $2m$ messages. The parent selection broadcast required at most $n$ messages in the case that each peer sends a request, and another $n$ messages for the parent acceptance message. This step required $2n + 2m$ messages.

5.5.3 Messages Involving Remaining Exchanges

The exchanges involving peers at or below depth two required approximately $4m$ messages in the worst case. Computing the depth required $m$ messages. Sending the child count required $m$ messages. Lastly, sending the exchange instructions and completion notification required $2m$ messages.

The exchanges involving peers at or above depth two required another $4m$ messages. Each peer sending its $f$-value and sum of its childrens’ $f$-values to its parent required $m$ messages each for a total of $2m$ messages. The exchange instruction messages required approximately another $2m$ messages.

So, we have $2n + \frac{3}{2}nm + 9m$ or $O(n + mn + m)$. 

62
Chapter 6

Minimizing FTHES Size

In this Chapter, we show that the maximum size of an FTHES as described in Chapter 3 is $2n - 3$. We also show that $n - 3$ of these exchanges are between strict local maximas within two hops of one another. Then, several experimental results show that in practice, the expected number of strict local maximas is much less than $n - 3$. Finally, through a reduction to the sum of subsets problem, we show that it is NP-hard to find an FTHES of minimum size.

6.1 FTHES Size

The size of an FTHES is equivalent to the number of exchanges in it. There are several benefits to minimizing the number of exchanges. It reduces the total number of transfers as well as the overhead required to track them. It
also reduces the average number of peers that any one peer exchanges with. We hypothesize that this increases the overall trust of an FTHES since peers have to trust fewer other peers with their data.

Now, we show the maximum size of an FTHES.

**Theorem 6.1.** For any graph $G = (V, E)$ for which the even parity and neighborhood sufficiency conditions hold, there exists an FTHES of size at most $2n - 3$.

**Proof of Theorem 6.1.** To prove Theorem 6.1, we will consider the exchanges described in the constructive proof of Theorem 3.1. The first part of this proof described exchanges between strict local maximas used to eliminate strict local maximas within two hops of one another. The third part of this proof described the exchanges in the ascending forest. We will start by showing that the number of exchanges between strict local maximas is no more than $n - 3$.

Clearly, the maximum number of exchanges between strict local maximas is bounded by the total number of strict local maximas.

**Lemma 6.1.** For any graph $G = (V, E)$ for which the neighborhood sufficiency and even parity conditions hold, there are at most $n - 2$ strict local maximas.

**Proof of Lemma 6.1.** For convenience, let $S$ be the set of strict local maximas in $G$. Now, assume that the number of strict local maximas in $G$ is $n - 1$. Let $p$ be the single non-strict local maxima. In this case, each of the $n - 1$
strict local maxima must only be connected to $p$. Were they connected to each other, they would be non-strict local maxima. Our restriction that $f(p) < \eta(p)$ does not hold for any $s \in S$ in this case since $f(p) < f(s)$, $\forall s \in S$.

If we assume that the maximum number of strict local maxima is $n - 2$, we can construct $G$ so that the neighborhood sufficiency condition holds. Figure 6.1 shows this case.

\[ \square \]

**Lemma 6.2.** For any graph $G = (V, E)$ for which the even parity and neighborhood sufficiency conditions hold, at most $n - 3$ exchanges are needed to reduce all strict local maxima within two hops of another.

**Proof of Lemma 6.2.** Each two-hop exchange added to the support $F$ in this step reduces at least one strict local maxima. Clearly, when $|S| = 1$, no strict local maxima is within two hops of another. Therefore, in the worst case,
$(n - 2) - 1 = n - 3$ exchanges would be added to $F$ in this part.

Now that we have shown that $n - 3$ exchanges were needed to remove all cases where a strict local maxima was within two hops of another, we will show that an additional $n$ exchanges is required to complete the FTHES. For convenience, let $A$ be an ascending forest. Notice that $V(A) = V(G)$.

The first exchange step involved vertices that were at or below depth two in $A$. Each exchange added to the support $F$ in this step satisfied at least one vertex.

The second exchange step involved the top three levels of $A$. Let $r$ be the root of an ascending tree in $A$. This step considered three cases.

The first case is when $f(r) < \eta_2(r) - 2$. In this case, exchanges were added to $F$ in the same way as in the first exchange step. Therefore, each exchange satisfied at least one vertex.

The second case is when $f(r) = \eta_2(r) - 2$. In this case, a single exchange was used to make $f(r) = \eta_2(r)$. This one exchange did not satisfy a vertex.

The last case to consider is when $f(r) = \eta_2(r)$. In this case, each vertex exchanged directly with $r$. The last of these exchanges satisfied two vertices (one being $r$), and the prior exchanges with $r$ satisfied exactly one vertex. This last exchange that satisfies two vertices cancels out the single exchange described in the second case above.

We have now shown that each exchange added to $F$ satisfies at least one vertex. This shows that a maximum of $n$ exchanges is needed to satisfy each vertex in $A$. 66
The support for our FTHES required $n - 3$ exchanges to eliminate cases where a strict local maxima was within two hops of another and $n$ exchanges to satisfy the vertices in $A$. We have therefore shown that a total of $2n - 3$ exchanges is needed for our FTHES, and this concludes our proof. □

6.2 Expected Number of Strict Local Maxima

We now discuss an experiment designed to show how many strict local maxima we expect to see in practice. This is important since $n - 3$ exchanges were needed to remove all cases where a strict local maxima was within two hops of another. In this experiment, we generated random graphs of varying sizes. Then, in each graph, we applied both a random and a Pareto distribution to assign $f$-values to each vertex.

To assign $f$-values, we chose to use the Pareto distribution [40] and a random distribution. The Pareto distribution is a power law distribution and effectively simulates the expected distribution of data that peers need to backup, which is that the majority of peers have a small amount of data to backup and a few have a large amount to backup (80-20 rule). For comparison, we also used a random distribution to assign the $f$-values. These two distributions are show in Figures 6.2 and 6.3.

Each graph was generated randomly for the given number of vertices and edges. In this experiment, we generated graphs with 128, 256, and 512
Figure 6.2: Pareto distributed $f$-values. The x-axis shows the different $f$-value sizes. The y-axis shows the number of peers having a given $f$-value.
Figure 6.3: Randomly distributed $f$-values. The $x$-axis shows the different $f$-value sizes. The $y$-axis shows the number of peers having a given $f$-value.
vertices. For each vertex count, we chose the number of edges following a logarithmic scale. In each case, we made sure that there were no isolated vertices.

The boundary cases for the number of edges in each graph were taken to be \( n/2 \) and \( n*(n-1)/2 \). For a graph to have no isolated vertices, there must be at least \( \lceil n/2 \rceil \) edges. Clearly, the maximum number of edges is seen in a complete graph with \( n*(n-1)/2 \) edges.

While these are the two extremes, they are not very interesting in practice. In both cases, there can be a maximum of 1 strict local maxima. When there are \( n/2 \) edges, a strict local maxima could exist only when \( n \) is odd and three vertices form a connected component. In a complete graph, a strict local maxima could exist only when a single vertex has the largest \( f \)-value in \( G \), globally.

Since the boundary cases were trivial, we took the number of edges to be the values between them on a logarithmic scale. For example, when \( n = 128 \), we generated a graphs having 128, 256, 512, 1024, 2048, and 4096 edges. Surprisingly, the distribution used to assign \( f \)-values seemed to have little effect on the average number of strict local maximas seen in each graph variation. These results are shown in Figures 6.4 and 6.5.

The results shown in Figures 6.4 and 6.5 indicate that the maximum number of strict local maximas is encountered when the number of edges is twice the number of vertices (\( m = n*2 \)). This makes sense given that the average degree of each vertex would be expected to be two, and the minimum
Figure 6.4: Average number of strict local maxima for a graph containing 256 vertices with f-values assigned via a random distribution. The x-axis shows the number of edges. The y-axis shows the average number of strict local maxima for each edge count.
Figure 6.5: Average number of strict local maximas for a graph containing 256 vertices with f-values assigned via a Pareto distribution. The x-axis shows the number of edges. The y-axis shows the average number of strict local maximas for each edge count.
degree of a strict local maxima is two. As the degree of each vertex increases, the chance that each vertex satisfies the neighborhood sufficiency condition increases as well.

6.3 Minimum FTHES Size - NP-Hard

Now, we show that the problem of finding an FTHES of minimum size is NP-hard through a reduction to the sum of subsets problem. Recall from Chapter 3 that the problem of finding an FTHES corresponds to the problem of finding a two-hop \( f \)-factor. To show that this problem is NP-hard, we will show that the problem of finding a two-hop \( f \)-factor of minimum size is NP-hard.

It is important to note that the \( f \)-sufficiency condition is relaxed in the presentation.

**Theorem 6.2.** The problem of finding an FTHES of size \( k \) is NP-hard.

*Proof of Theorem 6.2.* The problem of finding a two-hop \( f \)-factor of minimum size can be reduced from the sum of subsets problem. Let the input to the sum of subsets problem be a multiset \( A = \{a_0, \ldots, a_{n-1}\} \) of \( n \) positive integers and a sum \( S \). We construct a graph \( G = (V, E) \) with a vertex set \( V \) and an edge set \( E \) and map \( f \)-values to \( V \) as follows.

For each \( a_i \) in \( A \), define the following three vertices in \( G \): \( u_i, v_i, \) and \( w_i \) and assign their \( f \)-values as \( a_i, 0, \) and \( 0 \), respectively. Then, define two additional vertices, \( x_1 \) and \( x_2 \), and set their \( f \)-value to \( S \) and \( A - S \), respectively.
Figure 6.6: Reduction of the sum of subsets problem to the two-hop $f$-factor problem.

Connect each $v_i$ to $x_1$ with an edge and each $w_i$ to $x_2$ with an edge. Finally, connect each $u_i$ to each $v_i$ and each $w_i$. In doing so, $v_i$ becomes a bivalent vertex between $x_1$ and $u_i$ and each $w_i$ is a bivalent vertex between $x_2$ and $u_i$.

Clearly, this translation can be done in polynomial time and is shown in Figure 6.6.

The smallest possible two-hop $f$-factor in $G$ as constructed above is $n$ (the cardinality of the set $A$). Clearly, each $u_i$ has to exchange all of its data with either $x_1$ or $x_2$ or the solution would not be a minimum one. This means that there has to be a subset of the $u_i$’s whose $f$-values sum to $S$, which corresponds to the sum of subsets problem. \qed
Chapter 7

MyBook: Application of FTHES

In this Chapter, we describe an application called MyBook, which ultimately motivated the work described in this thesis. MyBook was designed and developed to facilitate the management, organization, and structuring of topic/project-focused resources. A key feature of the application is that it separates content from the structure. This feature is borrowed from the concept of a table of contents in a textbook, which points to data in the book but does not store any of it. In a distributed setting, MyBook is a collaborative tool that allows peers to share and contribute to those focused resources, which we call collaborative groups. We now describe collaborative groups, the MyBook application, and a potential integration of the FTHES into MyBook in the P2P model.
7.1 Collaborative Groups

Though some degree of collaboration is present in existing P2P technology, the importance of the relationship between each of the peers is not emphasized. In a file sharing system such as Kazaa, all peers are treated equally. We hypothesize that there are groups of peers with common interests in these systems and that allowing these groups to collaborate will lead to users finding more content with higher relevance amongst other benefits.

We call a group of peers in a P2P system that shares a common interest a collaborative group. An example of a collaborative group is the group of peers interested in Bach organ music. Typically, these peers may have a hard time finding content relevant to Bach organ music; however, if they join a collaborative group, they could contribute their own resources and consume relevant resources contributed by other peers interested in Bach organ music. All peers in a collaborative group are considered to be adjacent to each other - i.e., friends.

There are several benefits of collaborative groups beyond finding relevant content. Two of these benefits are directly related to the FTHES - trust and fairness. We make the assumption that peers in a collaborative group that work together to compile and make available resources relevant to a common interest have a higher degree of trust in each other than other potentially random peers in the system.

Peers may belong to multiple collaborative groups at the same time. For
example, a programmer may belong to a C# and a Java collaborative group. In this case, it is likely that there is some overlap between these groups. Additionally, peers in either group are interested in the common subject of computer programming. This could be considered a friend of a friend situation where a peer in the C# group may have a higher degree of trust for a peer in the Java group than a peer in a Brittney Spears group.

The collaborative group is a natural environment for the enforcement of fairness. We say enforce because fairness is not an inherent characteristic of the collaborations. Instead, we assume that the majority of the peers in the group are benevolent and willing to expel malevolent peers that either consume resources without contributing (free-riders) or contribute malicious content.

7.1.1 Structuring Collaborative Groups - MyBook (MBK)

The topic-focused content in collaborative groups will be managed by a novel XML data format/organizational object, called MyBook (MBK), which provides elements for compiling, structuring, and organizing topic/project-focused resources such as files, Internet hyperlinks, and text notes (summaries). As its name may imply, the MyBook is based in-part on the general concepts of a textbook. For hundreds of years, textbooks have been used as containers to compile, structure, and organized topic/project-focused information and this is exactly the purpose of the MBK, only in terms of electronic resources.
The basis of the MBK’s design is the generalization of the fact that textbooks can be viewed as a hierarchy of topics. In the textbook, this hierarchy is built from its parts, chapters, and sections and is reflected in its table of contents. Instead of considering each individual element in the textbook’s hierarchy of topics, the MBK abstracts them into a simple element called a topic. Just as each part, chapter, and section in a textbook is typically focused on a sub-topic of its parent element and contains information and possibly other sub-topics related to that sub-topic, each MBK topic is meant to be focused on a sub-topic of its parent topic and to contain information and possibly other topics. The MBK’s topic is recursive structure is described next.

The general structure of the MBK is shown in Figure 7.1. At the root of the MBK is the book element. The book element is meant to store informa-
tion related to a MBK as a whole such as its author’s information (authorId, signature), its identification number (id; our equivalent of an ISBN), and its version (version) to specify which revision of the MBK schema a MBK uses. The book element is also the entry point to a MBK’s topic hierarchy (topic) which is rooted at a MBK’s main focus.

Each topic element in a MBK contains information related to its parent or sub-topic. In the case of the first topic stored in the book element, the information is related to the MBK’s overall focus. The topic element stores a name or title (name) for the topic and an identification number (id). It can also store other topic elements (topics) related to its topic. Here we see that the topic element is recursive. Most importantly, the topic element stores the content element which is where the file pointers, Internet hyperlinks, and text notes or summaries are stored.

The content element stores Internet hyperlinks (favorites), files (files), and text notes or summaries (notes). Each favorites element stores a collection of favorite elements, each files element stores a collection of file elements, and each notes element stores a collection of note elements.

The favorite element stores information for an Internet hyperlink. This information includes the hyperlink’s name or title (name), its URL, and an identification number (id). The note element stores information for each note. This information includes the note’s name or title (name), its text data (text), and an identification number.

The file element stores information for a file. This information includes
a name or title (name), an identification number (id), the path or location of the file (path), and several properties for the file (properties-last modified timestamp, size in bytes, type). In the case of the MBK, the path value is either an absolute or relative path to a file stored in the local file system.

In its early stages of development, there was a great deal of conversation regarding the type of path to store in the path value for a file. Absolute file paths have the advantage of allowing the MBK to be moved freely on the local system where it was created without modifying the path value. The use of absolute file paths has the disadvantage of making it difficult to move the MBK between computers without modifying the path values or trying to replicate the files in the same locations on the target machine. The advantage and disadvantage of using relative paths are the opposite of those for using absolute paths. Here, the MBK cannot easily be moved on the local file system but can easily be transferred between machines.

The name value of each element is used to facilitate the visualization of the content. Each topic element’s name value is shown in the visualization of the topic hierarchy in the far left panel as well as in the visualization of the selected topic’s sub-topics in the top left panel. Similarly, the name values for each favorite, file, and note are displayed in the other panels respectively.

The id value of each element is meant to store a globally unique identifier (GUID). This value supports both the navigation of topics and manipulation of the XML structure of the MBK. Each id attribute is of the type ID. When using the DOM method of XML access, an index is built using the attributes
marked with the ID type. This provides the programmer with direct access to elements via their id value and greatly simplifies the process of traversing the XML structure (tree). Without the id values, the programmer would have little or no choice other than the potential for having to traverse the entire XML structure when performing modifications or retrieving information.

7.2 Compile, Organize, Visualize - MyBook

Graphical User Interface

Collaborative groups can be managed through the use of the MyBook graphical user interface. The MyBook GUI provides a user-friendly way to compile, organize, and visualize topic-focused content. The MBK topic hierarchy is easily created and maintained through intuitive menus. Files can be added to any topic in the hierarchy. A built-in web browser provides instant access to web-based resources and gives users the ability to bookmark relevant content. Finally, a notepad provides a free-form text editor that gives users the ability to add textual content to a topic, which ranges from simple notes to detailed descriptions (through the use of the rich text format RTF).

Figure 7.2 shows a screenshot of the MyBook graphical user interface.
Figure 7.2: The MyBook GUI - left panel contains the topic hierarchy; top left panel contains a breadcrumb trail of topics in a given path; top right panel contains file references; bottom left panel is a built-in web browser; bottom right panel allows editing of 0-many notes for a topic.
7.3 Proposed P2P Model

The MyBook application has been written and is freely available for use; however, the P2P model to host the collaborative groups (built around CBKs) has only been proposed. This proposed P2P model will host collaborative groups interested in collaborating not only to compile and share topic-focused content but to also preserve it via backup/archive. In this section, we describe several components of our proposed P2P model. These include an extension of the MBK format called CBK (collaborative MBK), a peer-interface based on the MyBook graphical user interface, and components for computing distributions of data using the FTHES, for tracking the data, and for monitoring the health of data in the network.

The visualization of the organization and structure of the content in each collaboration is a different approach to typical P2P file sharing. In most cases, P2P file sharing systems present content to peers by providing a keyword search interface. Peers enter keywords that are pertinent to the content which they are trying to find and query the system for that content. In cases where the keywords are insufficient or incorrect, the desired content may remain hidden from the peer.

7.3.1 Collaborative Book - CBK

To support collaborative groups in a P2P network, we extend the MBK to a slight variant called a collaborative book or CBK. An important difference
between the MBK and the CBK is that the CBK will only store pointers to files and Internet resources. The actual files in each collaboration will be stored/replicated throughout the P2P system in the leech space of peers and/or locally if a peer is storing a given file. This has the added benefit of greatly reducing the overall size of each CBK since they will contain simple XML text and not the actual content of the files to which they point. Certainly, the space needed to store the file path to an MP3 is considerably smaller than storing the MP3 itself in the CBK. Their small size will allow the potential for a substantial number CBKs to be stored throughout the P2P system or possibly on Internet servers depending on the chosen configuration.

Using the MBK as the basis for the CBK will precipitate a need to change the value of the path element. What value is stored in this element in the CBK depends on whether the model is implemented as a centralized or distributed P2P system. We have implemented a light-weight centralized P2P system in which we store a link to a web service which is used to track the location of the files used in the collaboration.

7.3.2 Peer Interface

Our peer interface, in conjunction with the CBK, will allow peers to easily view and navigate the content of a collaborative group and to easily upload or download content where desired. Consulting a textbook’s table of contents for the location a desired topic and information and flipping to the appropriate page or pages is replaced by consulting the topic of hierarchy of the
CBK/CBook and clicking it to view its content. Figure 7.3 shows our peer interface. The CBK’s version of the textbook’s table of contents is displayed in the far left panel. In this panel, peers will easily be able to navigate the topics of a CBook. The content in a textbook is typically in the form of text, figures, or tables. The content in a CBook will be in the form of files, Internet hyperlinks, and possibly text notes or summaries. Any sub-topics, files, Internet hyperlinks or text notes will be displayed in the remaining four panels. Peers will be able to download files by selecting them from the files panel. Peers will also be able to add files to the given collaboration from the files panel.

In addition to collaboration in the form of sharing resources, our peer interface could also allow direct interaction in the form of instant messaging between peers. Work done in [20] could be integrated into our peer interface and would provide a rich set of tools to allow peers to collaborate in a more direct sense of the word. This type of collaboration would be useful in any type of collaborative group but would be especially useful in a collaborative group that is centered around a team working on a project (i.e., project-focused).

7.3.3 Content Distribution - FTHES

In our proposed P2P model, content will be backed up through the use of FTHES. Backing the content up has several advantages. Most notably, in a P2P network, peers are expected to connect and disconnect at random
Figure 7.3: Proposed peer interface.
intervals, which is called churn. Without a backup, any time a peer disconnects, that peer’s content becomes unavailable to the other users in the collaborative group. A backup copy increases the likelihood of that content being available even when its owner is not. Even with a backup, any time a peer disconnects leaves the backups stored by that peer unavailable. To increase the availability of content, any number of replication protocols could be implemented including erasure codes [23].

In given intervals (e.g., days, weeks), each collaborative group will perform a backup of its content. To do so, each peer will report the amount of data that it has to be backed up to a central location. Recall that it is unlikely that an FTHES can be computed when considering only adjacent peers (friends). Therefore, we connect collaborative groups that are closely related given some similarity metric. This means that content from a collaborative group that cannot be exchanged within that group may be exchanged with peers in a related group, which is assumed to have a higher degree of trust. After an FTHES has been computed, the data is exchanged and a full backup has been created.

7.3.4 Content Tracking

Clearly, the location of each block of data in the system needs to be kept track of or the data will be lost. After an FTHES has been computed, the Content Tracker will record the location of each block of data being exchanged. When a peer wants to retrieve a backup of its content or consume another peer’s (in
a collaborative group) content, that peer need only provide the tracker with the ID (from the C-Book) of the file. From there, the tracker can lookup the locations of the blocks for that file and provide them to the requesting peer.

### 7.3.5 Monitoring

In the churning environment of a P2P system, it is expected that peers will be constantly connecting and disconnecting. When a peer disconnects, any blocks which it was storing are no longer available. This could possibly lead to the case where any number of files are rendered unavailable because there are no longer enough blocks to reconstitute them.

For this reason, each peer will need to monitor the health of its data that was exchanged with other peers. Peer’s whose exchanged data is in danger of becoming unavailable would form another smaller group within the collaborative group so that another FTHES could be computed to prevent loss. This could possibly result in some duplication so those peers would be allowed to delete the data previously exchanged with the now unavailable peers.
Chapter 8

Conclusions and Future Research

In this thesis, we have introduced a distributed backup model which addresses the issues of fairness and trust - two inherent issues of P2P networks. In this model, fairness was enhanced by requiring that each peer store exactly the same amount of data as it sends out for backup. Trust was enhanced by requiring that pairs of peers exchange equal-size blocks of data and by restricting the distribution of any data to within two hops of its source.

We then described a backup scheme, which we called the *fair two-hop exchange scheme* or FTHES, and showed that while an FTHES could be computing using existing $f$-factor algorithms, these algorithms were not easy to implement in practice. To support our FTHES, we introduced and proved a fundamental existence theorem. This theorem stated that an FTHES exists.
under our two fairly weak conditions - the even parity and neighborhood sufficiency conditions. We then showed that a near-fair solution could be obtained when the even parity condition was relaxed.

Next, we showed a linear time sequential algorithm and a translation of it to an efficient distributed algorithm. We then showed that a large portion of these algorithms was devoted to handling strict local maximas. With that in mind, we described several experimental results which showed the expected number of strict local maximas in practice to be roughly $\frac{1}{5}n$. To address situations where a restriction on the amount of data that each peer exchanges is important (e.g., erasure codes), we provided an edge-capacitated version of our sequential algorithm.

8.1 Future Research

There are several possible directions for future work. For all of the algorithms given, efficient implementations need to be written. The bound on the number of messages given in Chapter 5 was approximated and can and should be tightened.

In Chapter 6, we showed that our FTHES requires a support of size at most $2n - 2$. If the $n - 2$ exchanges in the support between strict local maximas could be eliminated, the support would be reduced to $n$. Some initial investigation shows that this may be possible in trees for which our even parity and neighborhood sufficiency conditions hold.
Conjecture 8.1. The size of the \( f \)-exchange set \( F \) is at most \( n \).

Another future research direction involves the simulation results given in Chapter 6. Through our simulations, we showed that the distribution of the \( f \)-values had very little impact on the number of strict local maximas. Future work includes additional experimentation to verify the results through the use of different graph generation and \( f \)-value distribution techniques.

A final future research direction is to use an FTHES implementation in a distributed network such as that generated from the Facebook social network. In fact, a backup solution based on our FTHES could be created and offered as an additional service that Facebook provides. Clearly, this would require additional work in the areas of security, privacy, among others.

8.2 FTHES: Considering Edge Capacities

Another interesting future research direction is to consider the FTHES when a capacity is assigned to each edge. Edge capacities could be considered such that the amount of data exchanged between two peers could be restricted. These restrictions could be tied to a wide-range of metrics including network latency, reputation, or redundancy/replication. Some work has already been done and is presented below.
8.2.1 Overview and Notation

Let $G = (V, E)$ be a graph, let $f : V \rightarrow \mathbb{Z}^*$ be a function that maps each $v \in V$ to an even non-negative integer $f(v)$, and let $c : E \rightarrow \mathbb{Z}^*$ be a function that maps each $uv \in E$ to a non-negative integer $c$ such that $0 \leq c \leq \min\{f(u), f(v)\}$. The neighborhood sufficiency condition holds for a peer $u$ if and only if $f(u) \leq \sum_{uv \in E} c(uv)$. Clearly, this implies that $G$ also satisfies the earlier definition of the neighborhood sufficiency condition ($f(u) \leq \sum_{uv \in E} f(v)$); however, as we will show, this algorithm must take care to not use any edge $uv$ more than $c(uv)$.

Instead of building an ascending forest as we did in the previous Chapters, we build a set of weighted trees called tree covers. We define a tree cover $T(V, E)$ to be a subgraph of $G$ such that $V(T) \subseteq V(G)$ and $E(T) \subseteq E(G)$. Let $w$ be the weight of $T$, i.e., the number of units that each peer in $T$ exchanges. To maintain the neighborhood sufficiency condition in each tree cover, $w \leq \min(c(uv))$, for all $uv \in E(T)$ must hold. In this algorithm, we assume $w = 2$ for each tree cover.

Let $\tau$ be the set of tree covers such that each $v \in V(G)$ is in exactly $f(v)/2$ tree covers.

8.2.2 Generate Tree Covers

The generation of the set of tree covers $\tau$ is straightforward. First, the largest $f$-value in the entire network is found. We call this $GLOBAL\_MAX_f$. Then,
we build a spanning forest $T$ that connects peers, when edge capacity is available, having $f =$GLOBAL_MAX$_f$. Next, the $f$-values and edge capacities of the peers and edges in the spanning forest and GLOBAL_MAX$_f$ are decremented by 2.

It is possible that simple trees (single peers) exist as components in $T$ for any GLOBAL_MAX$_f$. Let $p$ be a peer in a simple tree $t$, $t \in T$. As we will show later, the tree covers in $\tau$ are used to compute exchanges exactly as they were computed in Chapter 4. Clearly, $p$ has no other peer to exchange with in $t$ so we track this as “leftover” that will be addressed in a later step. We call this leftover $\ell$. In this case, $\ell(p)+ = 2$ for any $p$ in a simple tree.

Finally, each non-simple tree $t$, $t \in T$, is added to $\tau$.

### 8.2.3 Covering $\ell$

It is possible that a number of peers were forced to ignore (i.e., add to their $\ell(p)$) units that they need to exchange. Now, we consider the tree covers in $\tau$ and extend them such that any $p$ having $\ell(p) > 0$ becomes a member of $\ell(p)/2$ additional tree covers. In doing so, we must not affect any other peer.

There are two schemes for increasing the number of tree covers that some $p$ having $\ell(p) > 0$ belongs to.

1. “split” existing tree covers that $p$ belongs to

2. reconfigure tree covers that $p$ belongs to
Figure 8.1: Split a tree cover $t$ such that $p$ belongs to $\text{deg}_t(p) - 1$ more trees.

For the first scheme to work, $p$ must belong to at least one tree cover $T$ in which the degree of $p$ in $t$ is greater than 1 (i.e., $\text{deg}_t(p) > 1$). In this case, the tree cover $t$ can be split into at most $\text{deg}_t(p)$ tree covers. This splitting could result in $p$ belonging to an additional $\text{deg}_t(p) - 1$ tree covers.

An example of such a tree splitting is shown in Figure 8.1. Since $\text{deg}_t(p) = 3$ in $t$, $t$ can be split into 3 tree cover trees. Notice that the only peer affected by the split is $p$. The other peers in $T$ belong to a single tree cover regardless of how many times $t$ is split at $p$.

The second scheme involves reconfiguring tree covers that $p$ belongs to so that the resulting tree cover can be split at $p$ as described in the first scheme. It is only used when the first scheme can not yield enough additional tree covers so that $\ell(p) = 0$. Let $t$ be a tree cover that $p$ belongs to in which $\text{deg}_t(p) = 1$.

It is important to notice that if the first scheme was not sufficient, there must be at least one edge $pq$ such that $c'(pq) > 0$. In fact, $\ell(p) \leq \sum_{pq \in E(G)} c'(pq)$. 

94
These edges exist because they were not used because they would have created a cycle at some point when generating the forest covers. If no splitting was possible and no such edge(s) with remaining capacity existed, then we could assume that $p$ did not satisfy the neighborhood sufficiency condition in the first place, a contradiction.

Let $pq$ be an edge for which $c'(pq) > 0$. One of two cases must exist. The simplest case is that there is a tree cover $t \in \tau$ containing $q$ but not $p$. In this case, $pq$ can be added to $t$ to increase the number of tree covers containing $p$ by one. The second case is that there is a tree cover $t \in \tau$ which contains both $p$ and $q$ but not the edge $pq$. In this case, adding the edge $pq$ to $t$ forces $t$ to contain a cycle. If it did not create a cycle, the edge would have already been in $t$ as part of the spanning forest.

Now, we show how to handle the second case where we add $pq$ to $t$ and create a cycle, which must be broken. First, add the edge $pq$ to $E(t)$. Then, let $t'$ be a breadth-first traversal of $t$ starting at $p$. Starting at $p$ ensures that both the newly created edge $pq$ and $p$'s pre-existing edge are included in the traversal. Clearly, $deg_p(p) = 2$, which allows us to apply the first scheme to split $t'$ into two tree covers. Doing so increases the number of tree covers that $p$ belongs to by one while not affecting any of the other peers in $t$. The original $t$ is removed from $\tau$, and the two new tree covers are added to $\tau$. Finally, $\ell(p)$ and $c'(pq)$ are decreased by two. This technique can be repeated for $p$ until $\ell(p) = 0$ for any $p$ having $\ell(p) > 0$. 

95
8.2.4 Compute Exchanges in $\tau$

Recall that each $v \in V(G)$ belongs to exactly $f(v)$ tree covers in $\tau$, and that weight $w$ of each tree cover is two. For each $t \in \tau$, we can apply the ascending forest exchange techniques described in Chapter 4. In this case, the root of each $t$ can be any arbitrary peer since each peer is exchanging exactly two units.

8.2.5 Future Work

The worst case complexity of this algorithm is expected to be on order of $O(n^2 + m^2)$. An exact analysis is left as future work. Additionally, we leave the presentation of the algorithm itself as future work.
Bibliography


