Reorganization in Network Regions for Optimality and Fairness

by

Robert E. Beverly IV

B.S., Computer Engineering, Georgia Institute of Technology (1997)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2004 [September 2004]

Author IV
Department of Electrical Engineering and Computer Science
August 27, 2004

Certified by ....
Karen Sollins
Principal Research Scientist
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Reorganization in Network Regions for Optimality and Fairness

by

Robert E. Beverly IV

Submitted to the Department of Electrical Engineering and Computer Science
on August 27, 2004, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

This thesis proposes a reorganization algorithm, based on the region abstraction, to exploit the natural structure in overlays that stems from common interests. Nodes selfishly adapt their connectivity within the overlay in a distributed fashion such that the topology evolves to clusters of users with shared interests. Our architecture leverages the inherent heterogeneity of users and places within the system their incentives and ability to affect the network. As such, it is not dependent on the altruism of any other nodes in the system.

Of particular interest is the optimality and fairness of our design. We rigorously define ideal and fair networks and develop a continuum of optimality measures by which to evaluate our algorithm. Further, to evaluate our algorithm within a realistic context, validate assumptions and make design decisions, we capture data from a portion of a live file-sharing network. More importantly, we discover, name, quantify and solve several previously unrecognized subtle problems in a content-based self-organizing network as a direct result of simulations using the trace data.

We motivate our design by examining the dependence of existing systems on benevolent SuperPeers. Through simulation we find that the current architecture is highly dependent on the filtering capability and the willingness of the SuperPeer network to absorb the majority of the query burden. The remainder of the thesis is devoted to a world in which SuperPeers no longer exist or are untenable.

In our evaluation, we introduce four reasons for utility suboptimal self-reorganizing networks: anarchy (selfish behavior), indifference, myopia and ordering. We simulate the level of utility and happiness achieved in existing architectures. Then we systematically tear down implicit assumptions of altruism while showing the resulting negative impact on utility. From a selfish equilibrium, with much lower global utility, we show the ability of our algorithm to reorganize and restore the utility of individual nodes, and the system as a whole, to similar levels as realized in the SuperPeer network.

Simulation of our algorithm shows that it reaches the predicted optimal utility while providing fairness not realized in other systems. Further analysis includes an epsilon equilibrium model where we attempt to more accurately represent the actual reward function of nodes. We find that by employing such a model, over 60% of the nodes are connected. In addition, this model converges to a utility 34% greater than achieved in the SuperPeer network while making no assumptions on the benevolence of nodes or centralized organization.

Thesis Supervisor: Karen Sollins
Title: Principal Research Scientist

Contents

List of Figures

List of Tables

List of Algorithms

You are told a lot about your education, but some beautiful, sacred memory, preserved since childhood, is perhaps the best education of all.

- Fyodor Dostoevsky

The impossible often has a kind of integrity to it which the merely improbable lacks.

- Douglas Adams

Acknowledgments

This thesis would not have been possible without a colorful and varied cast of characters. In these acknowledgments I cannot hope to thank them properly.

First and foremost, I must thank my advisor, Dr. Karen Sollins, for patiently guiding me as I wandered in different directions and occasionally took tangential dives off the deep-end. I am very lucky to have her as a mentor. Karen's excitement, experience and novel perspectives continue to challenge and inspire my research.

Thanks to all of my former colleagues and coworkers from MCI: Randy Nicklas, Greg Miller, Kevin Thompson, Vint Cerf, k claffy and the entire highly talented vBNS bunch. I learned from each and every one of you and thank those who took me under their wing. They provided me with a unique opportunity to pursue that which was interesting and gain invaluable experience along the way.

I thank Dr. Richard Klein for his selfless care in mending my broken hand.

I am indebted to the members of the Advanced Network Architecture (ANA) and Networks and Mobile Systems (NMS) groups who showed me the ropes at MIT and steered me in the right direction. Steve Bauer and Peyman Faratin always lent a ready ear and perceptive feedback. Nick Feamster was a pleasure to work with. Thanks to John Wroclawski for his quick wit and insightful comments. Dina Katabi's knack for providing constructive criticism was equally matched by her ability to propose excellent solutions and research directions.

My officemate Mike Afergan was unequaled in his ability to engage me in argument, dispel the unreasonable and teach me what constitutes interesting research. I thank him for our many discussions and his friendship. Many ideas in this thesis originated from our conversations.

Finally, I dedicate this thesis to my parents who have provided unparalleled love and support during my studies and throughout my entire life. They are my heros; every child should be so lucky.

The cleverest of all, in my opinion, is the man who calls himself a fool at least once a month.

- Fyodor Dostoevsky

Chapter 1

Introduction

This thesis proposes an architectural mechanism to address the scalability, fairness and optimality issues at the heart of many overlay and distributed application networks. In particular, we make no assumptions as to the benevolence of users or nodes as is done in many such systems. Our architecture leverages the inherent heterogeneity of users and places *within* the system their incentives and ability to affect the network. We believe such architectures are important in increasingly competitive, and often hostile, environments such as the Internet.

By way of introducing our work, we detail trends in Internet usage and policy that motivate this research. Indeed, actual data captured from the Internet is used throughout and many design decisions are the result of analyzing how users behave in dynamic, complex systems. We briefly review overlay architectures in general and their weaknesses including fairness, optimality and the presumption of altruistic nodes. A summary of major contributions and an outline of the remainder of the thesis conclude this Section.

1.1 Motivation

The physical and logical structure of networks is vital to their scalability and robustness. Network overlays and distributed application frameworks are no exception. Two trends pose impending challenges to the logical construction of future overlays. First, the number of network elements is growing rapidly as devices become small and pervasive. Second, rather than being "simple" clients, nodes are increasingly both producers and consumers of data. The popularity of Peer-to-Peer (P2P) file sharing [38] and distributed computation systems, e.g. [39], provide compelling evidence of this transformation. However, the proliferation of P2P overlays and the resulting traffic load has driven many policy implications. Among these changes are service providers who cap the maximum aggregate traffic or traffic rate and organizations that block P2P systems to avoid the traffic charges and liability issues.

As the functionality traditionally provided by servers is distributed among all nodes in the network, it is useful to view every node as a small database or server. Rather than requiring support from the underlying network infrastructure, nodes dynamically organize themselves in an ad-hoc fashion to form logical overlays. By routing data through the overlay, program designers have been able to create applications that solve the distributed lookup problem [3] without assistance from the network or any centralized systems.

To date, the majority of overlay research has focused on algorithmic efficiency and purely technical issues, ignoring the policy and incentive problems that plague real systems. Current architectures overlook issues of fairness and assume the benevolence of a large set of users. The lack of fairness in the system causes: i) "free-riding," where nodes exploit others without contributing; ii) other nodes to be overburdened; and iii) lack of stability as nodes move or disconnect in an attempt to equalize the unfairness.

A second problem is suboptimal performance due to architectures that ignore the inherent node heterogeneity. The lack of optimality in the system causes: i) nodes to be burdened simply transiting data they have no interest in; and ii) unnecessary traffic and congestion.

Our research considers the formation, fairness, optimality and equilibria of these logical overlays. We propose an interest-based reorganization architecture to address the issues outlined above. We believe such architectures are important in increasingly competitive, and often hostile, environments such as the Internet.

1.2 Overlay Architectures and the Internet

We provide a detailed examination of prior work in overlays and topology formation in Chapter 2. However, this section briefly reviews the two large classes of overlay architectures and their current strengths and weaknesses as further motivation for our system.

Structured overlays [34, 47] tightly constrain node and content identifiers, which determine their locations in the topology, through a Distributed Hash Table (DHT) abstraction. Uniformly allocating content in the system, such that each node stores approximately the same amount of data, requires the use of a consistent hash function. The hash function determines which node stores each piece of content. For example, in Chord nodes and content are hashed to identities that are uniformly spread over an identifier "ring". Therefore nodes must store others' content, irrespective of their interest in it, to maintain the DHT invariants that guarantee correct lookups with bounded complexity. A nodes' incentive to store content that is not her own is based on an expectation that others will act similarly and store her content. Rather than make strong assumptions as to the altruistic nature of nodes in the network, we focus on unstructured networks which can form organically, allowing nodes to selfishly connect and affect their topology.

A second class of overlays is those which are unstructured. In unstructured overlays, nodes connect to other willing nodes. Since content may exist at any topological point in the overlay, queries are flooded with a limited horizon breadth first search. An inherent trade off exists between the likelihood of locating content in the system versus replication factor, i.e. the extent to which a piece of content is duplicated and query propagation distance, i.e. the query traffic overhead. Short of flooding queries to all members of the overlay, users have no guarantee against false negatives. The compromise between successful operation and query flooding limits the self-scaling properties which motivate distributed P2P systems.

To alleviate query overhead, Gnutella [17] and other systems implement a two-level hierarchy of leaves and "SuperPeers" [42]. SuperPeers are nodes with high-bandwidth and low-latency connectivity. Leaves connect to SuperPeers and SuperPeers interconnect with each other. The system makes use of a simplified Bloom filter to shield leaves from irrelevant query traffic. Each SuperPeer maintains a hash of file names stored on its leaves. Queries are flooded between SuperPeers, but only forwarded on to leaves if the hash indicates a possible query match.

While this architecture has several benefits and has allowed the continued operation and expansion of the Gnutella network, we believe it has several flaws based on the fact that it is dependent on strong assumptions about users' benevolence. First, it relies heavily on the existence and altruistic nature of the SuperPeers. Second, it treats all users equivalently, regardless of their interests, ignoring rather than exploiting the inherent heterogeneity of users. Third, while users have the ability to change their behavior, for example by moving around the system or tailoring their sharing preferences, the current architecture places these functions *outside* of the system.

This work presents an alternative architecture that brings the user's ability to affect the network and their incentives *within* the system. As such, our architecture is no longer dependent on the assumed altruism of the SuperPeers or any other nodes in the system.

1.3 Contributions

In this thesis, we argue that the region abstraction [44] provides novel insight into finding optimal logical network structures in large distributed content systems. The region architecture is a proposed construct in designing large scale, widely distributed and heterogenous networks. A region is an abstraction that provides a grouping and partitioning mechanism. Members of a region share or inherit a set of common invariants, may move between regions or be in multiple regions simultaneously.

We use the region abstraction as a basis for developing a logical reorganization architecture. At a very high-level, it is intuitively obvious that network elements may wish to logically reorganize in response to load, performance or failure scenarios if given the mechanisms to do so. We extend previous notions of interest-based locality [2, 45] to define regions and support our architecture.

The architecture we propose, and detail in subsequent Chapters, is a distributed mechanism whereby nodes explore and reorganize on the basis of interests. We define a utility function that captures interests and traffic load. In our design, nodes independently reorganize to maximize their local utility. We do not constrain the connections of any node; a node may add any number of neighbors that increase its utility. However nodes which are dissatisfied with a peer will disconnect from that peer, for example if the cost of connecting to that peer is higher than the benefit.

Network reorganization is an important fundamental concept that is well explored, particularly in altruistic, cooperative environments. Less well understood is the ability to provide both *optimality* and *fairness* in the presence of selfish nodes. We consider reorganization algorithms in domains of non-cooperative nodes that increase global utility and prevent free-riding thereby facilitating formation of an optimal and fair network.

To evaluate our algorithm within a realistic context, we capture data from a portion of the live Gnutella network and simulate various aspects of our algorithms. More importantly, we discover, name, quantify and solve several previously unrecognized subtle problems in a content-based self-organizing network as a direct result of using the trace data. We show, through simulations driven by our Internet trace data, the ability of our architecture to provide fairness while maintaining optimality.

The key contributions contained in this work are:

1. The formalization of ideality, fairness and optimality notions in P2P systems. We use these formalisms to evaluate our architecture.

2. An interest-based reorganization scheme. We perform simulations of our algorithms using

real-world work loads captured from a live portion of the Gnutella network. We provide extensive analysis assessing optimality and fairness and show the system achieving both.

3. An extensive trace-driven analysis of the relevant patterns of similarity and user behavior in P2P traffic. This both validates the assertion of prior work and provides a rich basis for our analysis.

4. A quantitative analysis of the dependence of systems such as Gnutella on the existence and altruism of SuperPeers, bringing a significantly new level of understanding to the problem.

5. Delineation, as a direct result of using real-world data, of several previously unrecognized problems in content-based self-organizing networks along with solutions and their quantitative analysis.

The remainder of this thesis is organized as follows. In Chapter 2 we present related work in overlays and distributed network formation, many of which this work builds upon. Chapter 3 describes our P2P data capture methodology and includes detailed analyses of locality and similarity present in the system. In addition, we discuss reasons why the current architecture is strained and provide further motivation for our work. Before reasoning about optimizing or reorganization, we first formally define ideality, fairness and optimality in the context of P2P systems in Chapter 4. The Chapter continues by detailing our interest-based reorganization design and implementation Chapter 5 begins by presenting details of the simulation methodology. We evaluate our design against existing architectures through extensive simulation and show the success of our approach. Finally, Chapter 6 concludes with a summary of major findings, contributions and suggestions for further research.

He always attributed to his critics a more profound comprehension than he had himself, and always expected from them something he did not himself see in the picture.

- Leo Tolstoy

Chapter 2

Background and Related Work

In this chapter we review the existing body of work on topics related to our thesis. We begin by presenting an overview of different network overlays, both research based and in popular use, including how they solve the distributed lookup problem. Next, we discuss other schemes that rely on locality within the overlay to provide context for our own work. Finally, we consider prior research that examines incentives, reorganization and selfish network formation.

2.1 Peer-to-Peer Overlays

To capitalize on the power and scaling properties of large distributed Peer-to-Peer (P2P) overlay systems, considerable research has focused on the distributed lookup problem [3]. A node seeking a piece of content must find, in a distributed fashion, which nodes store that content. Distributed lookup in P2P overlays holds the promise of self-scalability: the aggregate capacity of the network to service requests grows as a function of the number of nodes. Naturally the self-scalability property is limited by the altruism of network participants, a central theme of the work presented here.

The remarkable promise of self-scalability embodied by the P2P paradigm, has led to a wealth of theoretical, i.e. academic, and production systems. These systems fundamentally implement one of two approaches to building application layer overlays: unstructured and structured.

2.1.1 Unstructured

Unstructured systems are simple to build and maintain as evidenced by the relative success and popularity of working implementations on the Internet. Unstructured P2P overlays such as Gnutella [17], Kazaa [26] and FreeNet [11], despite their effectiveness and wide-spread use, are neither optimal nor fair [1] and provide excellent motivation for our work.[1] A central tenant of this thesis, shown in later Chapters, is the ability to retain the properties which make unstructured overlays successful while achieving optimality and fairness.

Unstructured systems can adapt, grow organically and reorganize dynamically. Nodes independently store their content without relying on any system-wide resources. Popular content, for instance a particular file, is often replicated by many nodes in the network. Nodes connect to one another with minimal constraints, flooding queries through the overlay with a limited horizon breadth first search. An inherent trade off exists between the likelihood of locating content in the system versus replication factor, i.e. the extent to which a piece of content is duplicated and query propagation distance, i.e. the query traffic overhead. Short of flooding queries to all members of the overlay, users have no guarantees against false negatives. The compromise between successful operation and query flooding limits the self-scaling properties which motivate distributed P2P systems.

To alleviate query overhead, Gnutella and other systems implement a two-level hierarchy of leaves and "SuperPeers" [42]. SuperPeers are nodes with high-bandwidth and low-latency connectivity. In contrast leaves are generally poorly connected and often behind firewalls. [2] Leaves use a bootstrap mechanism to connect in a pseudo-random fashion to SuperPeers and SuperPeers interconnect with each other. Each SuperPeer constrains the maximum number of leaves it will serve, so a leaf must repeatedly attempt connecting to many SuperPeers. In time, the leaf establishes a set, typically two to four, of stable connections into the SuperPeer network. Figure 2-1 illustrates an example section of the Gnutella network. [3]

The system makes use of a simplified Bloom filter to shield leaves from irrelevant query traffic. Upon connecting to a SuperPeer, a leaf sends a hash of the file names which it stores. Thus, each SuperPeer maintains a hash of file names stored on its leaves. Queries are flooded between SuperPeers, but only forwarded to leaves if the hash indicates a possible query match. The SuperPeer can know definitely that a leaf does not serve a piece of content, but cannot know with certainty that it does. Thus, queries may still be inadvertently sent to leaves who cannot provide an answer. We give

[1] We save precise definitions of optimality and fairness for Chapter 4. Intuitively however, optimal and fair networks can be said to maximize global welfare and ensure that nodes contribute in proportion to their derived benefit.

[2] The distinction between well and poorly connected is often fraught with difficulties. Most existing systems make use of ad-hoc mechanisms to make a determination as to which nodes should serve as SuperPeers. Well-connected nodes may elect not to serve as SuperPeers. We note that our architecture, presented later, is not required to make such a determination as nodes naturally evolve and emerge as central hubs.

[3] In reality, there is a third-class of node in Gnutella: a non-leaf, non-SuperPeer node. These nodes implement the older, flat topology of the original Gnutella architecture. They are allowed to connect in order to provide a migratory path to the hierarchical network. These nodes present some difficulty when collecting data which we detail in the next Chapter.

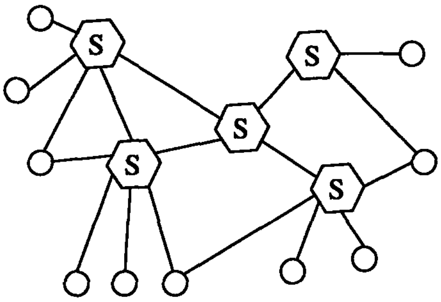

Figure 2-1: Gnutella Two-Level Hierarchy

details of Bloom filters in general and Gnutella filtering in particular in Section 4.5.

The two-level hierarchy has been lauded as a significant success, allowing the continual operation and expansion of the Gnutella network. However, the system depends on the *altruistic* nature of well-connected users to serve as SuperPeers and does not increase the chance of finding unpopular content. Singh et al. argue for placing the burden of maintaining SuperPeers on the service providers [41]. In contrast, our architecture turns the problem into a selfish distributed algorithm without depending on SuperPeers.

2.1.2 Structured

Despite the popularity of unstructured overlays, they lack correctness guarantees and are reliant on flooding. To address these shortcomings, often perceived as being too significant to overcome, many structured overlays [3] exist in the academic community. Chord and Tapestry [47, 48] are two popular examples; others include [19, 34, 37]. These structured overlays are generally realized through a Distributed Hash Table (DHT) abstraction. In contrast to unstructured systems, these overlays tightly constrain node and content identifiers, which determine their locations in the topology, through a Distributed Hash Table (DHT) abstraction. Structured overlays impose strong homogeneity where nodes maintain both a constant number of connections and probabilistically equivalent amounts of content. In most implementations, queries are guaranteed to be successful in $O(\log n)$ overlay hops if the content is present in the system. We compare structured and unstructured overlays in the next chapter.

Because structured networks enforce strict content placement such that nodes do not necessarily store their own content, the designers of structured networks rely on the benevolence of nodes to store others' content. Nodes are willing to store the content of other users in the system because they similarly rely on other nodes to store their own content. Thus, altruism is maintained through expectation.

Clearly, the storage fairness of structured networks can be easily gamed. Ngan et al. suggest an approach whereby nodes publish their resource consumption and contribution to the system [29]. By using a distributed auditing technique, nodes have an incentive to publish their resource records truthfully. However, this is just one of many fairness and incentive problems in structured overlays. The authors do not consider the problem of fairness as it relates to the bandwidth resources of nodes. Since content is allocated to nodes according to a consistent hash mechanism, a node may randomly be asked to store a piece of content that is very popular in order to be a part of the P2P system. Current methods rely on load-balancing mechanisms to prevent any node from saturating. The issue of fairness within structured networks is beginning to be recognized as a significant issue, particularly in proposed DHT services such as OpenHash [20].

Our work is similarly concerned with providing incentives such that users consume P2P resources in proportion to their resource contribution. However, we investigate the selfish formation of unstructured networks where nodes store their own content thereby eliminating fairness problems as they relate to file space resource consumption. Instead our distributed algorithm allows groups of users to interconnect such that the imposed query load is proportional to the benefit nodes derive from their connections.

2.1.3 Overlay Locality

Earlier work recognized the intuition that by preferentially connecting peers with similar interests together, one can minimize query flooding and thus optimize unstructured P2P systems. Keleher, et. al [21] put forth some of the first arguments for maintaining locality in distributed P2P systems.

Semantic Overlay Networks (SON) are proposed in [13] which develop a classification hierarchy such that a node belongs to one or more independent overlays and queries are tagged so they are sent to the right overlay. The difficulty with a rigid classification scheme is that its performance depends on defining criteria that classifies the data with sufficient granularity and requires users and nodes to classify their data.

The work of Sripanidkulchai, et. al [45] relies on the presence of interest-based locality to create interest based "shortcuts". Each peer builds a shortcut list of nodes that answered previous queries. To find content, a peer first queries the nodes on its shortcut list and floods the query only if querying the shortcut nodes is unsuccessful. We extend this idea to the network formation and use a utility model that considers fairness and cost.

2.1.4 Reputation Systems

A significant issue of concern is that of free-riders, nodes who download files without providing any value or who somehow misrepresent themselves to obtain benefit. Proposals, including this one, that attempt to mitigate the impact of free-riding generally rely on some type of reputation mechanism.

"Whitewashing" is the phenomenon whereby a malicious player (i.e. network node) circumvents reputation and repudiation algorithms by continually assuming new identities. Since nodes are assumed to be honest initially (otherwise the system could not bootstrap), malicious nodes exploit their initial credit by whitewashing. Note that identity in the system is typically not an IP address, but rather some system-specific identifier. While IP addresses provide an architectural mechanism for uniqueness, the proliferation of NAT and firewall devices does not preserve a one-to-one mapping between hosts and IP, i.e. violating the end-to-end principal.

In a self-reorganizing system, this certainly is a potential problem as there is a tradeoff between exploring a node and being exploited by that node. Free-riding and systems for establishing reputations are an issue of a significant amount of research [1, 23, 14]. However, reputations and truthfulness are beyond the scope of this paper.

2.2 Network Formation

This Section examines prior work in network formation, particularly as an optimization.

2.2.1 Reorganization

In [9] Chun outlines a network optimization game to optimize structured P2P networks. We use a similar framework to model our optimization game in the context of an unstructured P2P network.

Gia [7] is an unstructured P2P system that reorganizes in spirit similar to our design. Gia uses random walks for queries and biases the walk toward high-degree nodes. The additional insight in Gia is that these high-degree nodes can then become overloaded, so Gia implements a topology adaptation mechanism. Gia attempts to force high-degree nodes to be the ones with high-capacity and connects low-capacity nodes within short reach of high-capacity nodes. The simulation results demonstrate several orders of magnitude scalability improvement with the Gia scheme but does not address fairness or node attempting to game the system.

2.2.2 Selfish Formation

Recent research has examined the structure of selfishly constructed overlay networks [10]. The authors define a cost function that is proportional to distance and find node-degree distributions varying from node-degree to power-law. Similarly, through simulation we seek to understand the equilibria of graphs that are the product of our algorithm and measurement data. Because our utility is a function of interest locality, we expect to find significantly different results that complement this previous work.

Finally, the utility-based economic clubs proposed in [2] are closely related to our scheme. Their scheme also uses a utility function to drive each peer's reorganization strategy. While our simulation results of networks that include altruistic SuperPeers are similar, we are interested instead in purely selfish network. Thus, our focus is on network reorganization and formation in a world where SuperPeers no longer exist. Our work is complementary in that we examine the class of purely selfish and homogenous nodes derived from our trace measurements. In addition, we define different utility and reorganization models based on several previously unrecognized problems.

2.2.3 Regions

Our work is part of the regions project [44, 43] and develops methods of reorganization within and among network regions. The region architecture is a proposed construct in designing large scale, widely distributed and heterogenous networks. Regions are based on the notion that the Internet is an increasingly complex network of networks where elements are connected and interrelated by a set of common invariants. A region is thus a partition of the network where the member nodes share consistent control, state, policy or knowledge. The canonical example of a region is the set of nodes within an autonomous system (AS) [35]. The regions project seeks to provide an architectural mechanism with which to define and use regions.

We expand on earlier regions work [25] to understand how nodes participating in an overlay region can organize and segment in order to maximize their performance. The resulting, selfishly constructed islands of interest may be considered sub-regions, defining nodes with commonality that have reorganized together. In this work, we further codify the regions abstraction.

Explanations were advanced, but most of these were simply phrases which restated the problem in different words, along the same principles which had given the world "metal fatigue."

- Douglas Adams

Chapter 3

Analysis of P2P Data

The ability of a distributed reorganization algorithm to exploit interests depends on the ability to successfully drive a node to the proper region of the network. However, if a node's queries and files are sufficiently dissimilar, finding a sustainable position in the network for the node is tantamount to an impossible bartering problem.

Prior work has assumed the existence of high-degrees of locality within unstructured Peer-to-Peer (P2P) systems. Before considering reorganization algorithms based on presumptions of locality, we validate this assumption against real-world work loads. To perform the validation, we collect several days worth of data from a portion of a popular unstructured P2P network. Understanding the degree of locality in working systems allows us to better design a mechanisms to exploit that locality.

In this Chapter, we first present our data collection methodology and operational specifics of the Gnutella network relevant to gathering the data. We then give details of our collection program. Next we provide descriptive statistics on the data collected including the distribution of: connection duration, shared file count, query count, query rate and shared file size. As a basis for subsequent design decisions in future Chapters, we analyze the prevalence of file replication among nodes in our data set.

Finally, we ask to what extent, if any, interest-based locality can be leveraged in an architecture. We examine and quantify the existence of locality in our data set using algorithms and techniques adopted from the information retrieval community.

The data and results in this Chapter are tangential to our actual architecture, but vital in validating assumptions, making design choices and modeling real-world work loads.

3.1 Data Collection Methodology

For our study we gather data from a portion of the Gnutella network over a 72-hour period including approximately 1500 nodes. Our capture program, `grawl`, short for Gnutella-crawler, establishes itself as a SuperPeer on the Gnutella network and anonymously gathers queries and file lists from all nodes that connect to the host running our collector program. A significant fraction (approximately two-thirds) of peers refuse to answer directory queries, presumably for privacy reasons. Our analysis includes only those hosts which allowed listing of their shared files. While this eliminates many nodes from consideration, the duration of our data collection in conjunction with the probabilistic nature of node attachment ensures that our sample is reasonably representative of the true population. All IP addresses are anonymized to maintain the privacy of the Gnutella nodes `grawl` converses with. The data collected for this study is publicly available from the following URL: `http://ana.csail.mit.edu/rbeverly`.

We are primarily interested in the correlation between a node's queries and its existing content (i.e. shared file list). Queries do not contain the IP address of the originating peer, instead they contain a unique identifier from a 16-byte space randomly chosen by the node issuing the query. Nodes have no knowledge of where a query originated beyond knowing from which directly attached peer the query arrived. Presumably this design feature provides additional privacy against third-parties mapping queries to nodes and IP addresses.

Query routing in Gnutella is performed as follows. Nodes in the network maintain finite state on previous queries, mapping identifiers to the peer from which the query was received. On the basis of the identifier mappings, query hits are forwarded back toward the originator. Figure 3-1 illustrates the query response mechanism with an example. A node issues a query for key "ABC" and gives the query an identifier of "101." The query is flooded among the SuperPeers and sent to a node attached to S_5 that can answer the query. SuperPeer S_5 maintains a table that maps query identifiers to incoming interfaces and routes the reply to S_3. Similarly, S_3 sends the reply back to S_1. The arrow in the figure represents the query return path.

As mentioned in Chapter 2, a third-class of Gnutella nodes are those legacy nodes which do not implement the latest protocol supporting the two-level hierarchy. These nodes are allowed to connect in order to provide a migratory path from the old, flat Gnutella architecture. We also allow these nodes to connect and term them "non-leaf, non-SuperPeer" nodes.

In order to disambiguate queries received from peers that originated the request from queries

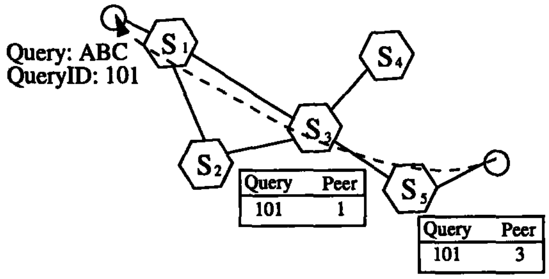

Figure 3-1: Gnutella Query/Response Mechanism

simply forwarded on behalf of other peers, grawl uses two techniques. Leaf nodes are guaranteed to have no children. As a SuperPeer, grawl records all queries from its leaf peers since these queries must be generated by that peer. Second, peers are required to increment the hop count and decrement the time-to-live (TTL) fields in the Gnutella packet header. Grawl records all queries from non-leaf, non-SuperPeer nodes with a hop count of one. When a peer disconnects, its query list is flushed to disk along with the connection duration.

In addition to recording the queries, grawl attempts to acquire the shared file list of each new peer that connects. The collector asks peers for their file list in two ways. First, grawl attempts an HTTP connection to the host as several clients provide an HTML list of files. Unfortunately many hosts are behind firewalls. If the HTTP method fails, the collector sends a special query message with a TTL of 1 and search of four spaces. This special query is honored by many clients to index all files the host is sharing. Despite the best efforts of these two methods, a significant fraction (approximately two-thirds) of peers refuse to answer these directory queries, presumably for privacy reasons. Our analysis includes only those hosts which allowed listing of their shared files. Table 3.1 summarizes the operation of the collector program by describing its actions taken in response to network events. Unless otherwise specified, grawl acts as a normal SuperPeer. [1]

We perform a tokenization procedure on the queries and file names. To tokenize we eliminate: i) non-alphanumerics; ii) stop-words: "it, she, of, avi, mpg," etc.; and iii) single character tokens. We do not use other normalization techniques such as stemming tokens [24] to their root words, e.g. "singing" → "sing". Although stemming methods could raise the match quality, the issue is orthogonal to our problem. Thus, our similarity measures are under-estimators of the true similarity. The resulting tokens are produced by separating on remaining white-space in the string. We assume tokenization in the remainder of this document.

[1] In fact, grawl is based on the freely available open-source Mutella [27] SuperPeer code.

Table 3.1: Data Collector Event/Action Summary

Event #	Description	`grawl` action
1	Leaf i connects	HTTP-get i's shared files. If get fails, goto 2.
2	HTTP-get of i fails	Send special "get all files" query with TTL=1 to i.
3	SuperPeer s connects	Allow up to 50 SuperPeer connections.
4	Receive query from leaf i	Record and timestamp query.
5	Receive query from non-leaf, non-SuperPeer i	If hop-count=1, record query.
6	Non-SuperPeer i disconnects	Anonymize IP, flush queries and connect time to disk.

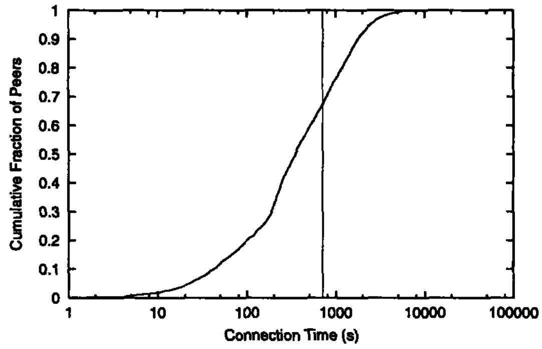

Figure 3-2: CDF of Node Connection Duration, Mean = 708 seconds

3.2 Data Descriptive Statistics

We now present descriptive statistics of the data captured and used in simulation which we refer to in later Chapters. For instance, we use the average query count and query rate to drive our calculation of the discount factor in our utility formula. Figure 3-2 presents the cumulative distribution of connection times our SuperPeer experienced during data collection. Similar to previous studies [18, 38], we see that the average connection time is approximately 11.8 minutes, a very short time. As few as 5% of the nodes were connected for more than an hour. 50% of the nodes connected for less than 349 seconds, approximately 6 minutes.

Such high degrees of connectivity churn are likely indicative of three factors. First, many nodes are free-riding, i.e. connecting only long enough to download a particular piece of content and then leaving the system. Second, many nodes are attempting to find SuperPeers that are nearby as measured by some latency mechanism. And third, many nodes are attempting to find content, but

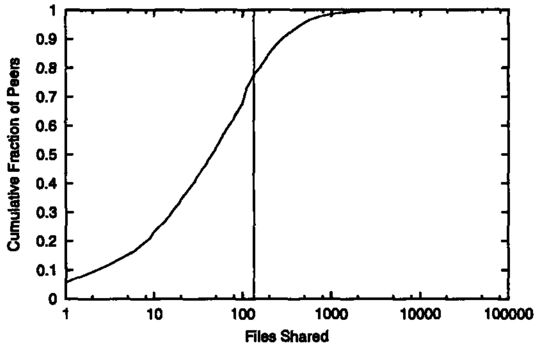

Figure 3-3: CDF of Node Shared File Count, Mean = 134 files

are unable to do so and are thus manually reorganizing.

In the next section we present our algorithm which places the reorganization within the system. We believe that our algorithm, as it takes incentives into account and prevents users that are connected for long durations from being taken advantage of, will lead to longer connection durations and hence greater stability. Indeed, in order for a node to reorganize such that it maintains connections to its preferred regions will take a finite amount of time providing a *disincentive* to disconnecting.

From capturing the list of file names and sizes, we gain insight into the properties of the content which nodes are willing to share. In Figure 3-3 we see that on average, hosts shared 134 files and only 20% offer fewer than 10 different pieces of content. Thus, even in a network where users have no incentive to provide files or answer the queries of other nodes, a significant proportion of those nodes that do share files share a fairly large number of files. [2] We use this fact as inspiration for our epsilon equilibrium model in Section 5.4. The mean number of queries per host was 11 as shown in Figure 3-4. Over 40% of the hosts in our data set sent a single query, again a likely indicator of churn in the system.

The total size of all files shared was almost 1 GB as seen in Figure 3-5. This leads to the observation that not all files should necessarily be considered equal. A file that is very large might provide greater utility than one that is small. We do not differentiate between files in this work, but our design could be supplemented by this intuition.

Finally, the average per-node query rate was approximately 0.012 queries per second, a value we use to determine an appropriate discount factor α in our simulations. Figure 3-6 displays the

[2] Recall that we must exclude approximately two-thirds of the nodes that refused to provide their list of files.

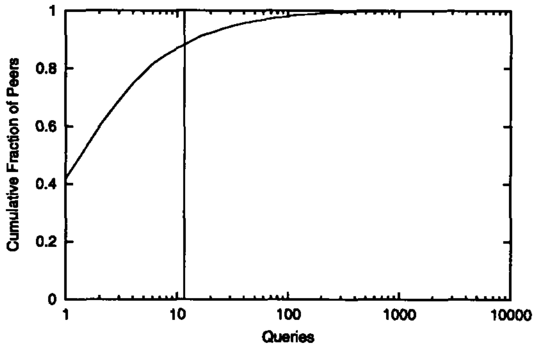

Figure 3-4: CDF of Node Query Count, Mean $\simeq$ 11 queries

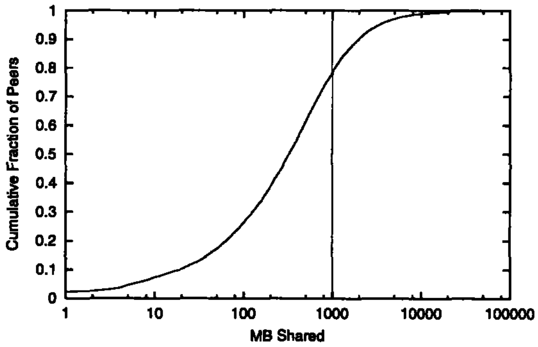

Figure 3-5: CDF of Node Share File Size, Mean = 997MB

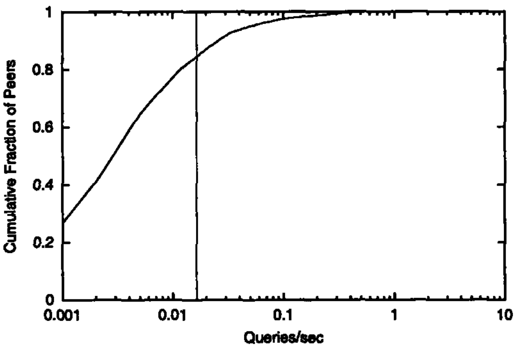

Figure 3-6: CDF of Node Query Rate, Mean ≃ 0.012 queries/second

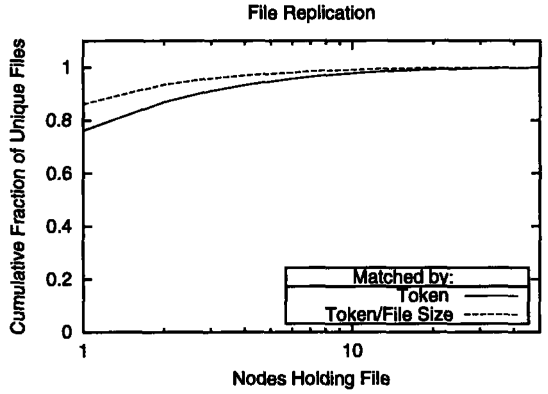

Figure 3-7: CDF of File Replication

distribution of query rates seen from all nodes.

3.3 Replication

An interesting feature of the data is understanding the extent to which file replication occurs in real systems. After tokenization, we find the number of nodes holding each unique file in the network as shown in Figure 3-7. Approximately 76% of the files are unique to a single node, indicating that 24% are duplicated at least once.

Two files may be different, for instance different formats of the same image, and still provide the same benefit for a user searching for that file. However, it may also be the case that replication is a more specific phenomenon where the file must be an exact replica. To determine the number of exact replicas of files, we change our definition of a unique file in the system to be the tuple of its file size in bytes and set of tokens. We see that approximately 86% of these unique files reside on a single node. Thus, 14% are duplicated at least once.

3.4 Interest-Based Locality

The ability to design a distributed interest-based reorganization algorithm depends on being able to successfully drive a node to the proper region of the network. For example, consider a node holding files that are all classical music, but issuing only queries for acid jazz. That node will have significant difficulty finding a portion of the network to which she can connect because nodes answering her queries will likely be unable to derive any benefit from her file store. In essence, this is a classic bartering problem and leads to our first question:

***Question #1:** "To what extent, if any, can interest-based locality be leveraged?"*

We are primarily interested in the correlation between a node's queries and its existing content, i.e. shared file list. In this Section, we seek to determine the similarity.

3.4.1 Similarity Metric

To assess the correlation between a Gnutella node's files and the queries it issues, we use two metrics from the information retrieval (IR) community. The Jaccard similarity [33] between a set of file tokens F and queries Q for node i is simply:

$$sim(F_i, Q_i) = \frac{|F_i \cap Q_i|}{|F_i \cup Q_i|} \tag{3.1}$$

We calculate the Jaccard similarity for the nodes in our data set where we successfully capture both the files and queries. Figure 3-8 displays the cumulative distribution of Jaccard similarity.

A more accurate and widely accepted similarity metric is cosine similarity, also known as the vector space model (VSM) [4]. Let the union of files and query tokens of node i be $U_i = F_i \cup Q_i$. For all nodes i in N total nodes from our study, let $U = \bigcup_i U_i$. Let $n_i = |U_i|$ for the files and queries of node i. We define two vectors of size n: $\vec{f_i}$ and $\vec{q_i}$ such that each element of the vector is a weight for a token in U. We use the Text Frequency, Inverse Document Frequency (TFIDF) formula for the vector weights:

$$\omega_{t,i} = (freq_{t,i}) log_2(\frac{N}{freq_t}) \tag{3.2}$$

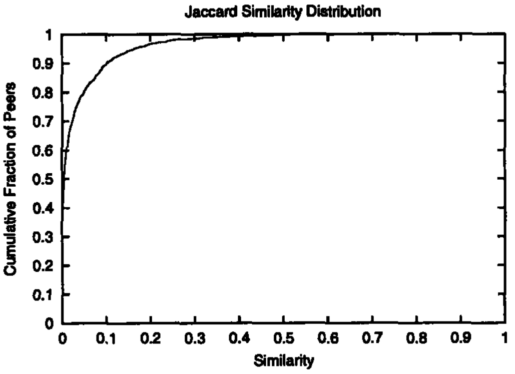

Figure 3-8: Jaccard Similarity CDF between Files and Queries

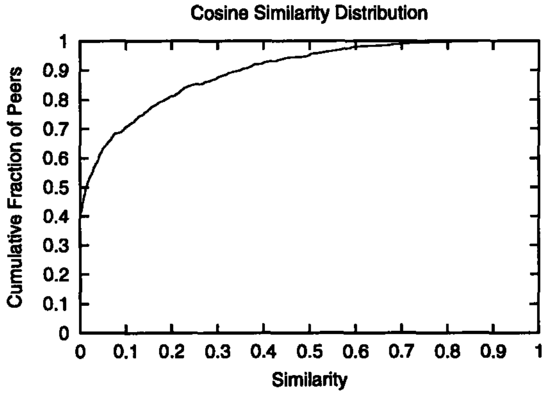

Figure 3-9: TFIDF-Weighted Normalized Cosine Similarity CDF between Files and Queries

Where $freq_{t,i}$ is the frequency of token t in U_i and $freq_t$ is the frequency of token t in U (all of F, Q). In order to cope with the varying lengths of each F_i, Q_i, we normalize by the length of the vectors. The similarity between the vectors is then the distance between them in an n-dimensional space. We compute the dot product of the length-normalized vectors to determine the cosine similarity between the files and queries of a node i:

$$sim(F_i, Q_i) = \frac{\vec{f_i} \cdot \vec{q_i}}{|\vec{f_i}||\vec{q_i}|} \tag{3.3}$$

Figure 3-9 displays the TFIDF weighted normalized cosine similarity between the files and queries of our data set. We see that 60% of the nodes we capture exhibit similarity between their file sets and queries. Our interest-based reorganization scheme, described in the next Chapter, is based on this encouraging result.

Conviction is worthless unless it is converted into conduct.

- Thomas Carlyle

Chapter 4

Design and Implementation

We present our design and implementation of a interest-based reorganization scheme in this Chapter. We focus on designing a system that converges to an optimal and fair structure, properties which we rigorously define. The system should be robust to malicious users who can misbehave or derive benefit from the system without contributing, i.e. "free-riding."

First we formally define ideal and fair networks within the context of Peer-to-Peer (P2P) overlays. To evaluate our design, we outline a continuum of optimality ranging from global to individual welfare maximizing networks. In addition, we show that enumerating all possible networks in order to find an optimal one, a task we perform in our subsequent algorithm evaluation, is exponentially hard.

Next we evaluate the dependence of the current architecture on the presence of SuperPeers. To support our argument, we show the use of Bloom filters in the system and analyze the false positive rate. As expected, we find that the system is highly dependent on the SuperPeers.

To take the first step toward a rational network equilibrium, we look at a SuperPeer model where leaves behave in an individually rational fashion. We then examine the potential for leaves to reorganize around a static SuperPeer network.

Finally we consider the problem at the root of our work, a world in which SuperPeers disappear or are no longer altruistic. We show a number of previously unrecognized problems in content-based self-organizing networks along with their quantitative analysis. Our utility function and reorganization algorithm are the result of careful analysis of these problems and the ability to properly model the system. Based on our trace data, we defend a choice of discount factor. We conclude by introducing four reasons for utility suboptimal self-reorganizing networks: anarchy (selfish behavior), indifference, myopia and ordering to give further basis for our subsequent analysis and results.

4.1 Defining Ideal and Fair Networks

Before we can begin to reason about optimization or reorganization we must first define metrics by which to evaluate our networks and algorithms. Here we provide our base definition for optimal and fair networks. The basic framework we consider, which we extend shortly, is:

- A set of nodes, N.
- An undirected graph $G = (V, E)$ of vertices (V) and edges (E) where each vertex is a node ($V = N$).
- For all $i \in N$, a set of files F_i and queries Q_i.
- A time-to-live (TTL), t, representing the query propagation depth.

4.1.1 Ideal

We define an ideal network as one where every query that can possibly be answered by any other node is successful. More formally, define $F_{i,j}(t)$ as the set of files available to node i as a result of maintaining an edge E connecting to j when queries propagate to a depth of t. For $t > 0$, we can express $F_{i,j}(t)$ by the recursive expression:

$$F_{i,j}(t) = F_j \cup \left(\bigcup_{k \in nbrs(j) - i}^{t>0} F_{j,k}(t-1) \right) \tag{4.1}$$

Thus $F_{i,j}(t)$ includes the files of the neighbor j to which i attaches as well as the files available to j, through all k of j's neighbors[1], with the TTL decremented. In other words, this expression gives the set of files determined by the union of all files available to i via nodes within its query reach as a result of the connection with neighbor j.

Let $S_{i,j}$ be the set of successful queries for node i as a result of connecting to node j:

$$S_{i,j} = Q_i \cap F_{i,j} \tag{4.2}$$

[1] k may or may not also be a neighbor of i depending on whether a cycle exists, but nothing prevents this from being the case. Since we are interested in the union of all files, multiple paths to the same node does not affect our definition

Let S_i be the set of successful queries for node i as a result of sending its queries Q_i to its complete neighbor set:

$$S_i = \bigcup_{j \in nbrs(i)} S_{i,j} \tag{4.3}$$

Define P_i as the set containing all possibly answerable queries of i, i.e. the number of queries answered as a result of connecting to every other node:

$$P_i = \bigcup_{j=0}^{|N|} S_{i,j} \tag{4.4}$$

An ideal network graph G construction then implies that the number of satisfied queries for all nodes $i \in G$ equals the maximum possible successful queries:

$$\frac{|S_i|}{|P_i|} = 1 \, \forall i \tag{4.5}$$

The canonical ideal network is a fully-meshed graph (a total of $\frac{1}{2}n(n-1)$ edges) such that every node has an undirected edge to every other node; many others will exist in practice.

Note that these definitions of the ideality of a network do not take into consideration identical context duplicated at more than one node. Thus, we require that each query be answered only by at least one other node for a match and give no additional benefit to multiple matches for a single query. In later sections, we consider the benefit of redundancy, i.e. the situation where query q of node i is for file f and nodes a and b both store f where $a \neq b \neq i$.

4.1.2 Fair

Fairness is the ability of the network to prevent nodes from gaining utility without contributing. A node that issues queries but does not provide any reciprocal benefit is a free-rider and should be disconnected from the network. Nodes provide benefit either by answering queries directly or by providing transit to other nodes who answer the query. The network should ensure the query success of each node is proportional to the query success it provides to its neighbors: [2]

$$\forall n \in N, \forall i \in nbrs(n) : s_n \propto s_{i,n} \tag{4.6}$$

[2] More precisely we are interested in maintaining *utility* fairness, but forego introducing utility until Section 4.6.2.

4.2 Defining Optimality

Optimality is then the balance between an ideal and fair network. We define a utility function that rewards ideal and fair networks while penalizing those with needless communication. The utility function is discussed in detail in Section 4.6.2, but abstractly we define $u_i(E_i, t)$ as the utility of a node i in the system when i has neighbors defined by E_i and queries are sent with a TTL of t. E_i, the edges of i, are defined by the graph G. The global utility of a graph G for TTL t is the aggregate utility of all individual nodes:

$$u(G,t) = \sum_{i=0}^{|N|} u_i(E_i, t) \tag{4.7}$$

In this section, we formally develop definitions for a continuum of optimality. At one end of the range are socially optimal networks which maximize the global welfare. Diametrically opposed are purely selfish networks where nodes maximize their individual their individual utilities. An optimal network of selfish nodes must provide no incentive for them to change their topology. Thus, we refer to selfish network optima as "equilibrium optimal." Finally, in the middle of the spectrum is risk-adverse altruistic optimality. Each of the three types of optimality are discussed next.

4.2.1 Socially Optimal

The socially optimal network is one that ignores individual user preference or happiness and instead maximizes the aggregate utility. For all possible graphs in the exponential set $\{G\}$, the socially optimal network is one that maximizes utility for a given TTL t:

$$G_{so}(t) = g \in \{G\} : u(g,t) \geq max[u(G-g,t)] \tag{4.8}$$

Socially optimal networks assume a centralized oracle planner is controlling node connections in order to provide the highest aggregate social utility. Because nodes cannot act autonomously in this model, individual nodes may be disenfranchised in order to provide benefit to other nodes if such a connection yields higher global utility.

To illustrate the fact that lowering one node's utility can increase the global utility by a larger amount than was lost by that node, we examine two possible networks. From a set of edges E, let $e_{i,j} \in E$ denote the undirected connection from node i to j. Consider graphs $g, h \in G$ identical except for g having an additional edge from i to j: $g = h \cup e_{i,j}$.

Then in the socially optimal model, the utility of g may be greater than the utility of h even if the utility for the single node i has gone down as a result of the connection from i to j. In this highly plausible situation, the global utility is greater with the connection from i to j because a third node k (possibly $k = j$, but not necessarily) contributes more global utility to the system than i detracts. Formally, $\exists i, j, k \in N, i \neq j, i \neq k$ such that:

$$\begin{aligned} & u_i(e_{i,j} \in E) < u_i(e_{i,j} \notin E) \\ & u_k(e_{i,j} \in E) > u_k(e_{i,j} \notin E) \\ & u(g) > u(h) \end{aligned}$$

The importance of this example is that some individual nodes may be "unhappy," or less happy than they might be, in order to maximize global welfare. The ratio between the socially optimal utility and the selfish utility is commonly known as the "Price of Anarchy" [30].

4.2.2 Risk-Adverse Altruistic Optimal

We consider a new class of optimality we term "Risk-Adverse Altruistic." Such networks fit between socially and equilibrium optimal. While these networks cannot be considered in equilibrium for purely selfish nodes, they provide a reasonable approximation of existing P2P networks. That is to say, we believe many P2P nodes are altruistic so long as they derive some, but not necessarily all, of the benefit of connecting to the system. We make only passing use of this model in our simulations and present it here primarily for completeness.

A risk-adverse altruistic optimal network is one where all nodes have non-negative utility (i.e. are "happy"). However, a node may not maximize its individual utility in order to provide higher global utility (i.e. be altruistic). We assume that nodes are not myopic and can implement a distributed algorithm to determine if their actions maximize the global utility.[3] Alternatively, we may again assume an oracle that serves as the central planner. Formally, the risk-adverse altruistic optimal network for a given t TTL is:

$$G_{rao}(t) = g \in \{G\} : u(g, t) \geq max[u(G - g, t)]; u_i(g, t) \geq 0 \, \forall i \tag{4.9}$$

[3] Unfortunately, assuming complete or semi-complete knowledge requires significant coordination, authentication and verification in a distributed system. Such a system is beyond the scope of this work and we use risk-adverse altruistic optimal only as a model by which to measure the performance of real systems.

4.2.3 Equilibrium Optimal

The final class of optimality we consider is equilibrium optimal. Such networks represent the highest global utility obtainable subject to the constraint that no node can disconnect one of its peers, i.e. remove an edge, and increase its individual utility. In other words, for the action space that includes dropping connections, the node cannot obtain higher utility by taking an action. Thus, the network is considered stable and in equilibrium. Formally, we define the equilibrium optimal network for a given t TTL as:

$$G_{eo}(t) = g \in \{G\} : u(g,t) \geq max[u(G-g,t)]; u_i(E_i,t) > u_i(E_i - e_{i,j},t)\ \forall i,\ \forall j \in nbrs(i) \tag{4.10}$$

4.3 Comparing Structured and Unstructured Overlays

Having defined ideality, fairness and optimality, we consider how to best architect a system to realize these goals. Again we turn to previous work for inspiration and a basis upon which to build. In this section we compare structured and unstructured overlays in the context of our goals.

The remarkable promise of self-scalability embodied by the P2P paradigm, has led to a wealth of theoretical and production systems. Here we compare structured and unstructured overlays in the context of fairness. Our design uses the region abstraction to realize a system with the most appealing properties of each.

Despite the power of structured overlays, Chawathe, et al. [7] succinctly identify several disadvantages in the context of the most popular P2P application: file sharing.

The first disadvantage stems from strict content placement in conjunction with the frequent membership changes characteristic of a realistic Internet P2P file sharing system [18, 36]. Uniformly allocating content in the system requires that, for each piece of content, a consistent hash function determines which node stores it (e.g., in Chord nodes and content are uniformly spread over an identifier "ring"). Therefore nodes must store others' content, irrespective of their interest in it, to maintain the DHT invariants that guarantee correct lookups with bounded complexity. Because of frequent node churn, the system incurs substantial communication and processing costs relocating content among nodes. A second drawback is that keyword searches are more relevant in file sharing networks than the exact match queries on which structured P2P systems rely. Keyword support in structured overlays is non-trivial. Finally, most queries are for "hay" rather than "needles". In other words, the majority of queries are for popular content which is, by extension, well replicated. By destroying the natural interest structure, a DHT is optimized for a uniformly distributed traffic load, an unrealistic assumption in production networks.

Unlike structured networks which have substantial maintenance overhead, unstructured systems are simple to build and maintain. However, the compromise between successful operation and query flooding limits the self-scaling properties which motivate distributed P2P systems.

Selecting an appropriate overlay depends on the intended application and neither structured nor unstructured systems are ideal. Unstructured overlays are appealing due to their low maintenance overhead, replication of popular content and ability to search on wildcard queries (i.e. "hay" not "needles"). However these same systems exhibit unattractive properties: no correctness guarantees, large amounts of query traffic and lack of fairness (i.e. "free-rider" syndrome). On the basis of these observations, we choose to base our design on unstructured overlays.

Our thesis is the following. There exists a natural structure in P2P systems, and regions in general, that stems from common interests. If nodes adapt their connectivity within the overlay to optimize for common interests, the overlay will evolve to clusters of users who have strong shared interests. These are connected with longer links that convey information about second level preferences. With the addition of utility-based trust mechanisms, the topology evolves toward increased efficiency and reduced congestion while pruning misbehaving nodes and free-riders. In other words, the system provides a closer realization of network optimality.

4.4 Reorganization Algorithm Design

Informally, the design of our system is as follows. A node entering the overlay uses a bootstrap mechanism to locate a set of initial peers as in existing P2P designs. Nodes attempt to maximize their utility by exploring the benefit of maintaining different connections in the overlay. Thus nodes are faced with a classic exploration versus exploitation problem.

Nodes selfishly attempt to maximize their local utility which is simply a benefit minus cost function we provide later. A neighbor peer that provides a query result increases the utility of maintaining a connection to that peer. Similarly, nodes lose utility as their neighbors issue queries and thereby increase the traffic load. Note that the utility of a peer providing a query response increases regardless of whether or not that peer is the node providing the file. The node may simply

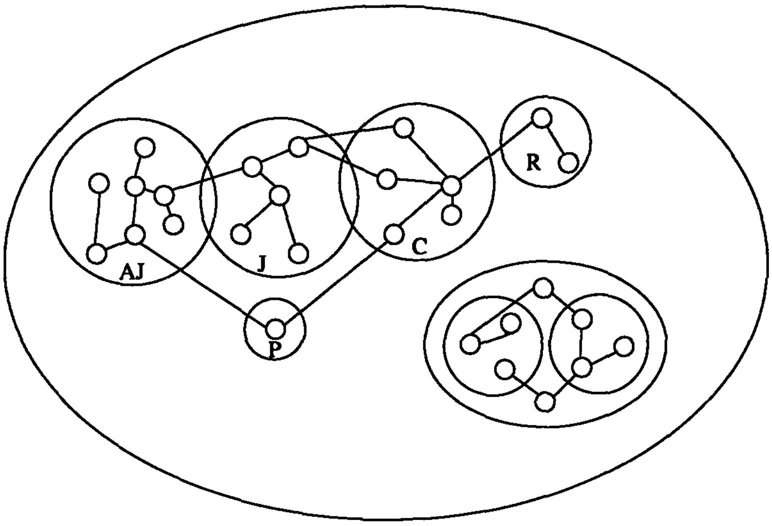

Figure 4-1: Interest Regions

be on the reverse path of the query response. In this manner nodes can provide both direct and indirect utility that may be either a positive or negative contribution. If a node forwards a query to its neighbors who do not have the file, it cannot provide an answer to the original querier. Thus nodes with dissimilar interests will tend to disconnect while those with similar interests will connect.

By not imposing a hierarchy, the system captures hidden correlations between groups of users. For instance, aficionados of acid jazz may also be avid programmers. The system will cluster acid jazz nodes in close proximity to nodes with source code. We graphically show interest regions of a hypothetical network in Figure 4-1. Acid Jazz (AJ) nodes are in a region connected to the Jazz (J) region. A second level preference is shown by the node in a region (P) which joins Acid Jazz to a region of programmers offering Code (C). This Figure illustrates that regions may exist within other regions and regions need not necessarily be connected.

To evolve, nodes in the system continually explore new links. The goal of the evolution algorithm is to eliminate nodes that only serve to relay messages, freeing system resources to improve system performance and scalability. We punish bad neighbors by disconnecting the link to them. In summary, our design provides the following benefits:

- Nodes in the overlay reorganize as a function of the queries they observe rather than relying on an explicit classification hierarchy.[4] By not imposing a hierarchy, the system captures hidden correlations between groups of users.

[4] We could use a large genre classification strategy as part of the bootstrap mechanism to improve convergence time, but do not consider it in this work.

- A utility function to model the derived benefit of each peer. The system provides fairness by ensuring that no incentive exists to misbehave.

- The system's global utility is increasing. While the topology may converge to a local optimum, it is continually optimizing toward an improved state, one that cannot be worse than Gnutella's random connection strategy.

- Fairness and optimality. Higher query success rates are achieved even with a lower time-to-live (TTL) and hence less query traffic. Free-riders are disconnected.

The utility function is a critical decision in our algorithm design and should model reality as closely as possible. However the utility function may require a very large number of variables to completely describe a particular node's preferences. Regardless, the utility is clearly a benefit minus cost function. Subsequent Sections describe our utility function in greater detail and analyze the simulation sensitivity to parameter changes in the function. The inability to accurately model the payoffs for a large set of independent players in a system motives the epsilon equilibrium analysis in Section 5.4.

4.5 Bloom Filters

Before analyzing the systems' dependence on SuperPeers, we provide background on Bloom filters [5] and their use in existing P2P networks. We then examine the Bloom filter false positive rate as derived from our data set.

Bloom filters provide a compact method of representing element membership. Consider a set E of n elements: $e_1, ..., e_n$. To construct a bloom filter, choose k independent hash functions: $h_1, ...h_k$ where each hash function produces a result in the range $[0, m)$. Create an empty boolean vector B of length m: $|B| = m$. For each element in the set, run each of the k hash functions to produce mk values. Each of the mk values sets a bit in the bloom filter vector B. For the j'th hash function, bits $h_j(e_i)\forall i$ will be set to true. Calculation of the bloom filter representation B of the element set E is given in Algorithm 4.1.

To determine if an element e is a member of B, check that $B[h_j(e)] = 1 \ \forall j$. Algorithm 4.2 formally states the membership query.

Bloom filters are appealing because they have the ability to represent element membership in a fixed size. For instance, Gnutella hosts use a simplified Bloom filter where $k = 1$ and $m = 64$ (B

```
for $i = 1$ to $m$ do
  $B[i] = 0$
for $i = 1$ to $n$ do
  for $j = 1$ to $k$ do
    $B[h_j(e_i)] = 1$
return($B$)
```

Algorithm 4.1: Bloom Filter Digest

```
for $j = 1$ to $k$ do
  if $B[h_j(e)] = 0$ then
    return(false)
  $B[h_j(e_i)] = 1$
return(true)
```

Algorithm 4.2: Bloom Membership Query

is a 64k-bit vector) to represent their list of files. Each file name is hashed to a value in the range $[0, 65535)$.

Bloom filters can provide false positives, indicating membership presence when the element is not actually in the set. However, there are no false negatives; a Bloom filter membership query that indicates the element is missing is always correct. The probability of a false positive depends on the number of elements, hash functions and size of the Bloom filter vector. The false positive probability rate is given by:

$$(1 - e^{\frac{-kn}{m}})^k \tag{4.11}$$

In Gnutella, with a single hash function, the false positive rate is:

$$1 - e^{\frac{-n}{m}} \tag{4.12}$$

As shown in Chapter 3, the average number of files shared by hosts in our dataset was 134. The expected false positive rate is: $1 - e^{\frac{-134}{65535}} \simeq 0.2\%$.

Within the context of P2P, Bloom filters offer an additional benefit: protecting the privacy of a node's shared file list. It is impossible to recreate the set of keys that generated the bloom filter. An exhaustive search of the key space would lead to far too many false positives. Thus, Bloom filters provide an elegant mechanism to both reduce communication cost and allow leaves to send a privatized digest of their file lists.

4.6 Dependence on SuperPeers

Having validated the potential for self-reorganization in Chapter 3 and outlined the advantages of unstructured overlays in the context of our goals in this Chapter, we examine the question of how best to incorporate reorganization into an architecture. We will consider reorganization in both homogenous and heterogenous networks, i.e. those with and without the two-level hierarchy of SuperPeers and leaves. We consider three potential architectures:

1. SuperPeers without Self-Reorganization: This represents the current architecture, what Gnutella uses today.

2. SuperPeers with Self-Reorganization: This would be an "augmented" Gnutella using the concepts of self-reorganization. For example, different SuperPeers could serve as "cluster leaders" for various interests, much as there are bulletin boards, etc. today.

3. Self-Reorganization Alone: This represents a world in which either SuperPeers no longer exist or we chose not to employ them.

To guide our decision, we examine the importance of SuperPeers in our dataset and thus the dependence of the system on the SuperPeers. An obvious additional question is "What would happen if the SuperPeers just went away?" While this is more at the heart of our argument, it is a difficult question to answer scientifically since it requires us to make many assumptions as to how the network would function. Regardless of the assumptions, it is clear that without these hubs, the network would suffer significantly. Since the primary role of SuperPeers is to filter and limit the queries, we ask the question:

***Question #2:** "What would happen to the system if the SuperPeers did not filter the messages?"*

4.6.1 The Base Model

To answer this first question, we turn to our dataset. The data presents us with an underlying framework for the network. Upon this framework we build a model which we use to drive our simulations. Here we extend the earlier framework, which we will build upon further in later sections:

- A set of user nodes, N, each corresponding to real users in the trace

- A set of SuperPeers, S

Table 4.1: SuperPeer Topology Construction Parameters

Parameter	Value	Description
C_a	10	Max Number SuperPeer Leaves
C_s	5	Max Number SuperPeer to SuperPeer Connections
C_l	2	Leaf to SuperPeer Connections

- A non-directed graph $G = (V, E)$ where $V = N \cup S$
- For all $i \in N$, a set of files F_i and queries Q_i corresponding to the files and queries observed for all non-SuperPeer player i's in the trace.

Recall that our data ignores the structure of the Gnutella network it was captured from. Instead, the data simply represents nodes and their corresponding files and queries. We create the network of leaves and SuperPeers as follows. Gnutella specifies a number of network properties we maintain: the number of connections leaves attempt to maintain with SuperPeers ($C_l = 2$); the number of leaf connections SuperPeers are willing to maintain ($C_a = 10$); and the number of SuperPeer to SuperPeer connections ($C_s = 5$). These parameters are summarized in Table 4.1. For N leaves, we create $\frac{NC_l}{C_a}$ SuperPeers. SuperPeers are randomly connected to form a d-regular graph using the method described in detail in Section 5.1.1. Each leaf is randomly connected to C_s different SuperPeers, subject to the constraint that a SuperPeer accepts at most C_a leaves.

Given this model, a run of the system in our simulator is the following four step process:

1. Each node sends all of its queries.
2. According to the TTL, the queries are flooded through the SuperPeer network.
3. According to the TTL and SuperPeer Bloom filtering, queries are propagated to the leaves.
4. Total query traffic and the number of successful query matches are recorded.

Figure 4-2 presents the results of this simulation, plotting aggregate number of queries against query TTL for a world with and without filtering The y-axis on the left of the graph shows the aggregate number of messages in the network without filtering while the y-axis on the right shows the same measure with filtering. In all simulation runs, the network topology is static. Clearly, the loss of filtering would have a significantly negative impact on the system, increasing query load by a factor of approximately 400!

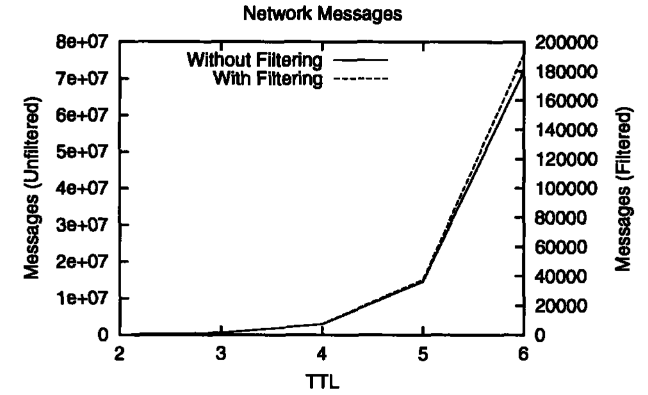

Figure 4-2: Queries Received by Leaves in SuperPeer Network vs. TTL

Observation #1: *The scalability of the current Gnutella system is highly dependent on the continued existence and functioning of the SuperPeers.*

4.6.2 An Individually Rational Model

One problem hidden in Figure 4-2 is that query message load not only causes scalability problems but may also cause nodes to reconsider whether or not they are happy being connected to the network at all. Nodes can consume considerable CPU and bandwidth resources processing incoming queries. It is reasonable to expect users whose queries are not being successfully answered to leave the network, particularly if remaining connected burdens the user.

Clearly, one way to reduce query-induced load is by decreasing the query TTL, further localizing query propagation. However, doing so directly decreases the query success rate.

To analyze the intertwined effects of the query load and success rate for a user, we enhance our model. Define an execution function $RUN(G, F, Q)$ which simulates the system produces a set of tuples (L_i, M_i) where:

- L_i is the aggregate load (in queries per second) received by node i in graph G. The induced load of a node j is calculated by taking the total number of queries issued by j in the trace and dividing by the total amount of time that it was connected in the trace.

- M_i is the total number of query matches obtained by node i from issuing queries Q_i through its links in graph G. In terms of our earlier definitions, the number of query matches is the size of the successful query match set: $M_i = |S_i|$.

We define a traditional benefit minus cost utility function that captures interests and load. Our benefit function is the square root of a node's query matches (M_i). Its convexity represents diminishing marginal return for additional matches. The cost of supporting the neighbor set is given by the linear function $g(x) = \alpha x$. We discuss choosing an appropriate value of α in Section 4.9.5. Define the utility of node i as:

$$u_i(G, M_i, L_i) = \sqrt{M_i} - \alpha L_i, \ \forall i \in N \tag{4.13}$$

Thus, the utility for i maintaining connections to its neighbor set is a convex function of the number of successful queries as a result of that neighbor set minus a linear function on the number of queries i has to process on behalf of its neighbors. An obvious question is whether this utility function, whose terms admittedly have inconsistent units, is representative of a user's preferences. We discuss this and examine the assumptions built into the utility function in the next Section.

This model provides the basis of an *individually rational* system. Nodes:

- Leave the system if they receive less than zero utility.
- Have no ability to affect the topology of the system.

We evaluate the SuperPeer network under the base model and the individually rational model in Chapter 5. The results of our simulations show the dramatic impact of Bloom filtering on the overall system utility. The key reason for the reduction in system utility without filtering is that a significant number of the nodes leave the system. Thus,

***Observation #2:** The sustainability of the current Gnutella system is highly dependent on the continued existence of the SuperPeers.*

4.7 Modeling Assumptions

Before proceeding further, we take a brief diversion to discuss modeling assumptions we make and their implications.

A key simplifying assumption made throughout is with respect to the temporal nature of the system. A real P2P overlay network is highly dynamic with nodes entering and leaving, gaining new content (e.g. downloading) and issuing queries over time. To lend tractability to a complex problem, we do not examine temporal aspects of the system. Instead, the nodes, files and queries are taken as a fixed set at one point in time. This allows us to focus on topology and reorganization issues. To properly model a temporal file-sharing overlay would require gathering data from a much larger section of the Gnutella network for a longer duration. We operate in this mode both to simplify the problem for initial analysis and since a dataset of a period on the order of weeks makes the analysis rather unwieldy. While we recognize the limitations of this mode, and discuss possible future work that includes time in Chapter 6, we feel that the effect of this assumption is especially minimal in those experiments where the network topology is fixed.

Our utility function includes a several assumptions. First, the benefit of query matches has diminishing marginal return, i.e. $\sqrt{M_i}$ in the utility function. Second, we assume that the cost of the incoming queries scales linearly with the load. Third, we assume that all users have the same utility function. Fourth, for our simulations we must select a value of α. We do not presume to be able to determine a "correct" utility function, but rather use it as a tool for making directional comparisons. We consider how to best choose a value for α in Section 4.9.5. The inability to model a large set of user preferences accurately motivates our investigation of an epsilon equilibrium in Section 5.4. Thus, we feel we are justified in our use of the utility function and assumptions presented here.

A third assumption is how to score multiple responses that a query can receive. In calculating M_i, we might count multiple hits from the source (i.e., found along different paths) as multiple matches. Or, the nodes could count only multiple hits for the same file from different sources as multiple matches. For example, in Figure 4-3, a node issues two queries: f_1 and f_2. While there are two paths to the node which stores f_1, and hence two possible query response paths (a query response may come to S_1 from both S_2 and S_3, each of which is forwarded to the original querier leaf) there is only a single match for f_1. Ignoring multiple responses from the same node prevents nodes from building dense graphs in order to maximize utility. However, some amount of path redundancy can be beneficial. In contrast, there are two distinct copies of f_2, thus benefit is derived from both as the query responses originate from different sources. We examine scoring multiple responses in Section 4.8.2.

4.8 SuperPeers with Reorganization Architecture

We have presented a model for evaluating and simulating the current Gnutella P2P overlay architecture. We next turn our attention to the second potential architecture, an augmented Gnutella-like

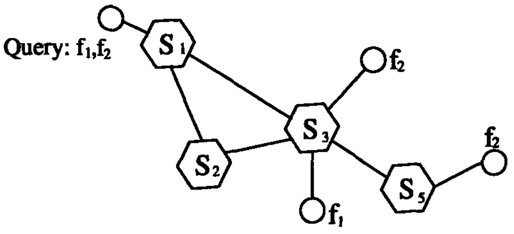

Figure 4-3: Redundancy and Calculating Query Matches

system where leaves reorganize around the SuperPeer network.[5]

4.8.1 Reorganization Model

Algorithm 4.3 specifies a naive per-round reorganization process for leaves in a SuperPeer network. Leaves attempt to find a SuperPeer they are not already attached to that is willing to accept new connections. Each leaf propagates its queries. Leaves receive queries from their SuperPeers when the SuperPeer's Bloom filter indicates a match. After all queries are propagated, nodes determine the utility of their SuperPeer connections and perform a drop step. Let $u_{i,j}$ be i's utility derived from maintaining a connection to node j.[6] We use the simple drop rule of disconnecting nodes that contribute negative utility.

```
$\{SP\}$ = set of SuperPeers
while $j$ busy do
  $j \leftarrow rand(SP - \{nbrs(i)\})$
Connect to $j$
Send and Receive Queries
for all $k \in nbrs_i$ do
  if $(u_{i,k} < 0)$ drop $k$
```

Algorithm 4.3: $leafReorg(i)$: Leaf i Reorganization

In this and later Sections, we employ a small facetious network to guide our design decisions or illustrate important observations. We construct a simple network consisting of 6 leaves and 10 SuperPeers. The files and queries of leaves 0, 1 and 2 were chosen such that these three nodes had strongly similar interests. Leaves 3 and 4 were created in the same contrived fashion but with a different set of shared interests. Finally leaf 5 has files and queries dissimilar from all of the other leaves.

[5] Some P2P systems rely on latency measurements to drive a leaf's selection of SuperPeer connections. In this sense, they too reorganize. Our reorganization is instead driven by fairness and optimality as previously described.

[6] We later show that utilities cannot be measured in isolation, but rather must be taken across coalitions of neighbors.

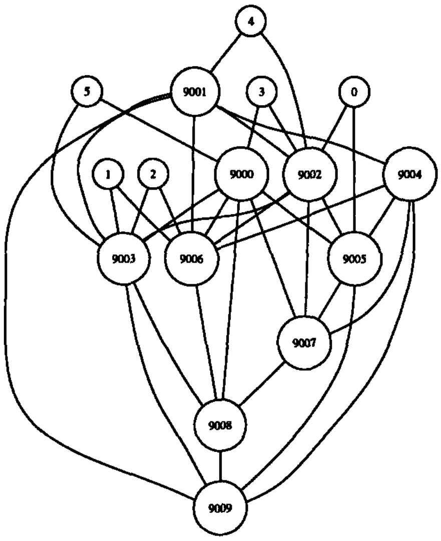

Figure 4-4: Example 10 Leaf, 6 SuperPeer Network Topology

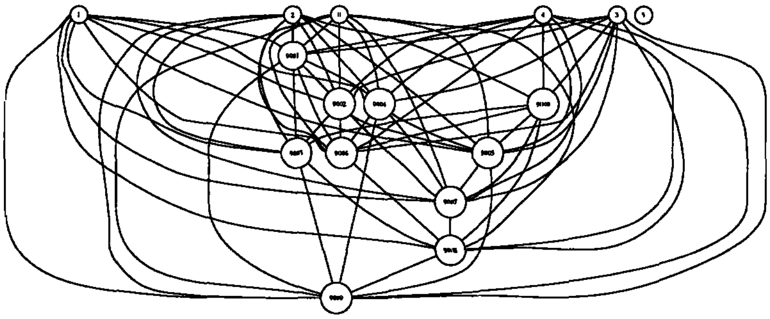

Figure 4-5: Leaf Reorganization within SuperPeer Network, $\alpha = 0.1$, $ttl = 2$

Using the same procedure as in Section 4.6.1, the simulator randomly interconnects the 10 SuperPeers (labeled 9000 through 9009 to clearly distinguish them from leaves) and then connects the leaves randomly to 2 distinct SuperPeers. In each of the simulations that follow, the simulation begins with this same initial topology, shown in Figure 4-4.

4.8.2 Simulation

The utility of the initial network for $\alpha = 0.1$ and a TTL of 2 is 17 if each SuperPeer has the bloom filter digest of its leaf children. If we allow the network to reorganize according to our algorithm, we see that utility quickly converges to 242. The resulting topology is shown in Figure 4-5.

In this dense graph, nodes 0, 1 and 2 have tried to connect to each other via many different SuperPeers. In fact, each SuperPeer is filled with the maximum number of leaf connections. The maximum number of leaves is five in this simulation and bounds the utility. Nodes with strongly similar interests gain utility with multiple connections up to the point that the utility imbalance, i.e. load, on a neighbor, in conjunction with α causes one peer to drop the connection. In this simple example, the TTL is 2 so the load effects are localized around a single SuperPeer and the low $\alpha = 0.1$ allows as many connections as the SuperPeers will support. Thus, we see the two groups strongly connected while node 5, the node which can only contribute load, is completely disconnected.

A second invocation of the simulation yields a slightly different utility result of 252. In this example, the order in which nodes discover each other via different SuperPeers influences the equilibrium result.[7] Since each SuperPeer maintains at most five leaf connections, once the SuperPeer is "full", other nodes cannot attach and explore the additional utility of the connection. If the nodes have a positive utility for the SuperPeer connection, the SuperPeer will never have any new connection slots. The leaf connection count in combination with the utility disbalance between a pair of nodes attached to the same SuperPeer is the main influence on the equilibrium stability seen in the example.

When nodes are risk-adverse altruistic (Section 4.2.2), where each node stops exploring once its utility is positive, the network quickly converges to a utility of 27. Nodes 0, 1 and 2 cluster together as do 3 and 4. Node 5 maintains a connection to every SuperPeer that does not connect leaves 0, 1, 2, 3 or 4. Since the utility of connecting to a SuperPeer that connects the other leaves is negative, 5 will always disconnect. Similarly, since 5 is indifferent to connecting to SuperPeers that don't have other leaves, it maintains connections to all of them. We note that while nodes acting in this semi-selfish manner allow the network to quickly reach equilibrium, the global utility is far short of that in the previous two examples.

Figure 4-6 shows the utility of the preceding three examples as a function of simulation round. The first two lines show the two simulation invocations where $\alpha = 0.1$ and TTL is 2. The third line shows the utility of the network in the risk-adverse altruistic model. Our simulations lead to the following two observations:

Observation #3: *Nodes should not derive additional utility from multiple paths to the same source node holding a particular piece of content.*

Observation #4: *The artificial limits on SuperPeer leaf connections can limit the final utility.*

[7] We discuss the ordering problem in detail in Section 4.11.4.

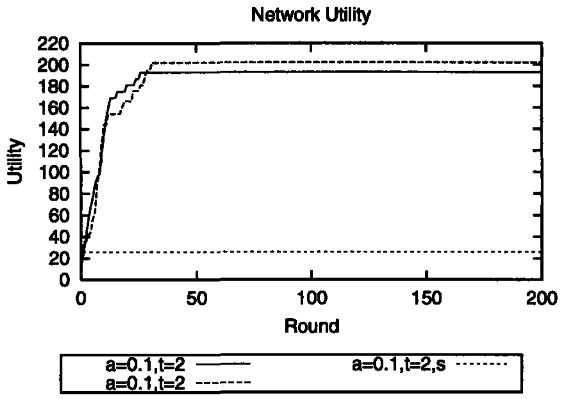

Figure 4-6: Network Utility vs. Reorganization Simulation Round in SuperPeer Network

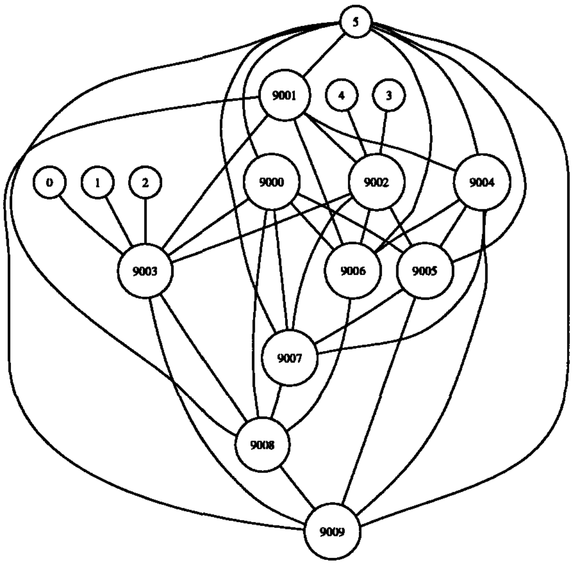

Figure 4-7: Effect of Changing α on Reorganized Topology, $\alpha = 0.25$, $ttl = 2$

4.8.3 Sensitivity to Alpha

We next explore the effects of changing α, the discount factor applied to query loads when nodes compute their local utility. Again, we start with the initial topology of Figure 4-4. The resulting topology for $\alpha = 0.25$ is shown in Figure 4-7.

Because of the increased cost of query load with $\alpha = 0.25$, nodes 3 and 4 cluster together around a single SuperPeer while 0, 1 and 2 cluster around multiple SuperPeers. In contrast to the earlier examples, 3 and 4 are relegated to their own cluster since the increased cost of load from SuperPeers that connect the other cluster forces the disconnection. Because there are unfull SuperPeers, the utility reaches a point where it oscillates between several utilities around 100 at equilibrium. The

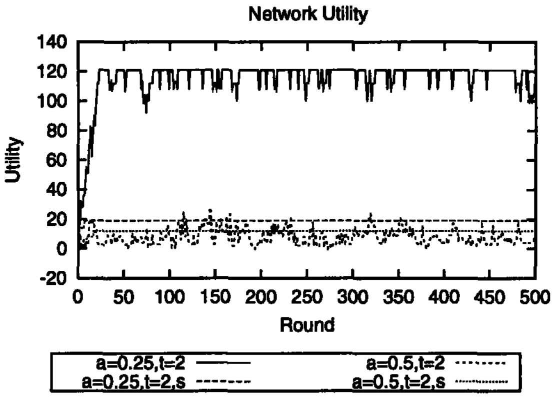

Figure 4-8: Network Utility vs. Round for Different α in SuperPeer Network

results of changing α are given in Figure 4-8. Thus, we make the following intuitive observation:

Observation #5: *The discount factor, α, affects the resulting reorganized topology and utility.*

We discuss selecting an appropriate value for α in Section 4.9.5. Our results and observations lead to our first conclusion:

Conclusion #1: *Given Observations 1, 2, 3 and 4, we decide not to include SuperPeers in our proposed architecture.*

4.9 The Full Model

With the background of the previous Sections, including partial models to build the proper intuition, we now present a full model for selfish, interest-based reorganization within a network of homogenous nodes (i.e. without explicit SuperPeers). We make the following refinements to earlier models:

- *Equilibrium:* As before, we allow nodes to have a disconnect action as part of their strategy. A node may disconnect any of its existing links and possibly be fully disconnected (for a utility of zero). Thus, it is the case that two nodes connected by a link in the overlay must desire for that edge to exist.

- *Reorganization:* Nodes reorganize independently in each round of simulation. Beginning with an initial topology, nodes explore by connecting to other nodes in the system. As before and per our assumptions in Section 4.7, nodes issue all of their queries and receive all possible responses in every round. On the basis of the responses, and the load from peers, nodes determine which members of their current peers they wish to retain and those they wish to disconnect.

- *Free-Riding, Truthfulness and Reputations:* A significant issue is that of free-riders, nodes who download files without providing any value or who somehow misrepresent themselves to obtain benefit. In a self-reorganizing system, this certainly is a potential problem as there is a tradeoff between exploring a node and being exploited by that node. Free-riding and systems for establishing reputations are an issue of a significant amount of research [1, 14] but beyond the scope of this paper. For the purposes of our simulation the issue is mostly moot as we ignore temporal aspects and effectively compress our trace data into a single period. However, we do note that communities and community structure provide a natural means of addressing the issue of reputations.

- *Number of Rounds:* In a real dynamic system nodes will be continually joining and leaving the system and hence the network will never be stable. Even in our simulations with a fixed population of nodes as derived from our traces, the sheer size of the network implies that it will take a significantly long time to stabilize. Therefore in our experiments, unless otherwise noted, simulations run for 10,000 rounds. If we assume 10 seconds per round, this corresponds to approximately 27 simulated hours.

4.9.1 Coalitions

In developing our reorganization scheme, we find that decision making is a key component with several subtle aspects. On the basis of prior experience exploring and the current set of connections, nodes must decide which subset of connections to keep. A naive approach evaluates links individually and forms per-link opinions toward a decision. However, this approach ignores the ability of multiple links to provide the same files. Thus, the utility contribution of an individual link is not important but rather its ***unique contribution.*** This is depicted in Figure 4-9.

In this Figure there is a set of four nodes, A, B, C and D, and their constituent files. We also show the load each of the nodes exerts on A. Consider A's decision process. A must decide which subset of its current peers it wishes to keep at each round. Assume A is trying to satisfy queries for files F_1, F_2, F_3 and F_4.

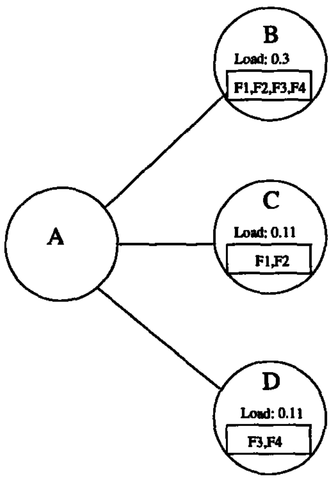

Figure 4-9: Network Topology Demonstrating Potential Coalitions and their Importance. The existence of B and C makes A undesirable.

If A is initially connected to B, all of A's queries are satisfied giving A a utility of $\sqrt{4} - 10(0.3) = 1$. However, if instead A were connected to C and D, A would also be able to satisfy its queries, but with a higher aggregate utility of $\sqrt{4} - 10(0.11 + 0.11) = 1.8$. To provide nodes with the ability to properly determine their best action, we borrow from Shapley to introduce the notion of coalitions [40]. See e.g. [22] for a more modern treatment of coalitions.

From our example, we see that it is impossible to score the links completely in isolation but rather the node must examine the potential coalitions. This is a potential downside since there are a combinatorial number of sets. In practice, the computational load on a given node is not significant since it will likely be considering only a small number of links. However, this does make the network topology dependent on ordering, an interesting observation that we discuss in Section 4.11.4.

We assume that when a node is indifferent between two coalitions, it will select the larger one in the hope that by doing so, it can aid global utility while not deviating from a selfish strategy.

4.9.2 Bullies

One issue related to the exploration problem is that if nodes eliminated undesirable peers every period, the system stability and potential for exploration would be significantly diminished. For a TTL > 1, when a node receives a query, it passes it on to all its neighbors. Thus, if a node connects to an undesirable peer, one with a high query load and low number of desired files, it may cause the peers already connected to it to drop since they too will suffer the higher query load. We term these

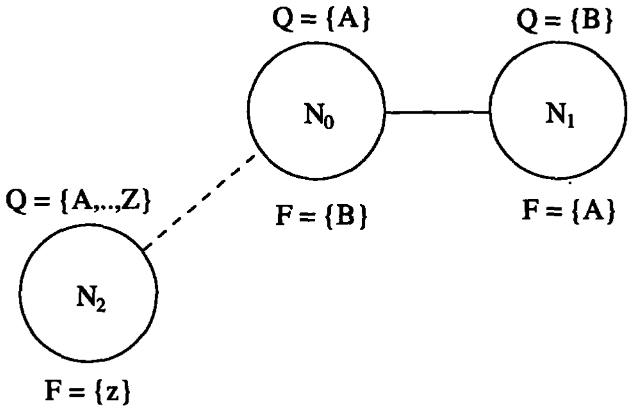

Figure 4-10: Bully Players in the System

undesirable nodes *bullies*.

For example, Figure 4-10 shows the effect of a bully player in the system. Assume nodes N_0 and N_1 are connected and no other connections exist in the network. N_0 and N_1 are in equilibrium, since both nodes are deriving utility from connecting to each other and they do not wish to deviate from their current coalition. Now consider a bully node, N_2 that has a large number of queries and a set of files that is either small or of minimal use to other players in the system. As N_2 explores, it might choose to connect with N_0. By connecting to N_0, the queries of N_2 are forwarded along to N_1, upsetting the equilibrium balance that N_0 and N_1 were in. Thus N_2 is acting as a bully in the system. If N_1 disconnects from N_0 as a result, before N_0 can disconnect the bully N_2, then the global utility and stability of the system will decrease as a result.

4.9.3 Hysteresis

In order to overcome this difficulty and to promote exploration, we implement a simple hysteresis in the nodes. Nodes maintain per-node happiness which, in combination with a simulated annealing strategy [31], drives reorganization decisions. The goal, in the context of bullies for example, is for a node to drop another node with some probability based on how happy or unhappy it is with it over the periods in which the two nodes have interacted.

Simulated annealing has the attractive property that instead of continually taking a selfish choice that can lead a node to a local minimum, nodes make decisions based on probabilities that exponentially decay. Algorithm 4.4 shows the boolean decision a node i makes for choices involving a second node j based on i's notion of happiness with j.

Each node maintains an notion of happiness, *happy*, with every other node in the system. $happy_i$ is a non-negative integer. In each round, when a node j is part of the optimal coalition for node i, $happy_i[j] = h_i[j] + 1$. Conversely, if a node is not part of the optimal coalition, i decreases its happiness with that node. At each decision point, the probability of a node taking an action X that involves j is given by: $Pr[X] = e^{-0.14 happy_i[j]}$. Thus, if $happy_i[j] = 0$, i will drop j with probability 1. If $h_i[j] = 15$, there is only a 12% chance that i will drop j during the drop phase. A node for which i has never interacted with previously is given an initial happiness of 15. Thus, nodes are initially considered good. $k = 0.14$ was picked to give a 50% decay after 5 time steps. Solving $0.5 = e^{5k}$ gives $k = -0.139$

```
$k = 0.14$
$r \leftarrow [0, 1)$
$p \leftarrow e^{-k(happy[j])}$
**if** $r \leq p$ **then**
   return($true$)
return($false$)
```

Algorithm 4.4: $makeMove(j)$: i Decision Making Based on j

4.9.4 Algorithm

In each round, the peers use the $makeMove$ algorithm to determine whether to accept a new connection or drop members of a coalition. By using simulated annealing, nodes quiesce as their utility converges to an optimum.

Algorithm 4.5 specifies the distributed reorganization algorithm. Each node randomly selects a node other than itself and its neighbors to connect with. After sending and receiving queries, the node evaluates its local utility for each coalition in the neighbor set and performs a drop step.

```
$\{N\}$ = set of all nodes
Connect to $rand(N - i - nbrs(i))$
Send and Receive Queries
**for all** coalitions $k \in nbrs(i)$ **do**
   $c \leftarrow BestCoalition$
**for all** nodes $j \in nbrs(i) - c$ **do**
   $happy[j] \leftarrow happy[j] - 1$
**for all** nodes $j \in c$ **do**
   $happy[j] \leftarrow happy[j] + 1$
```

Algorithm 4.5: $reorg(i)$: Node i Reorganization

Algorithm 4.6 provides pseudo-code for the drop algorithm.

```
if $makeMove(j) = true$ then
    drop $j$
```

Algorithm 4.6: $drop(i, j)$: Node i Peer Dropping

4.9.5 Choosing Alpha

Across our data set, we see that the average number of possible query matches per node is approximately 3. The mean number of queries per node is around 11. And each node's average query rate is 0.02 queries per second. Let k be the number of neighbors a node has on average. An average node can support:

$$\sqrt{M} - \alpha k^{ttl}(0.02) \geq 0$$

For $k = 3$ and $TTL = 2$, we find $\alpha = 9.6$. Thus, we choose $\alpha = 10$ for the experiments in this work unless otherwise noted.

Our final observation is the sensitivity of our results to α in the cost function. By increasing α, we create networks a greater number of smaller clusters and a larger proportion of fully disconnected nodes. Similarly, decreasing α produces networks with fewer larger clusters. While our results are sensitive to α, and in fact we achieve a much higher query success rate with larger α, the relative difference we demonstrate still holds.

4.10 Finding Optimal Graphs

In this section, we describe our technique for finding the "optimal" network structure. Understanding the ideal network that maximizes global utility provides a basis with which to compare the selfish algorithm. Unfortunately, an exhaustive search of all possible graphs is exponentially hard.

Theorem 4.10.1 *For $n = |V|$ nodes, the number of possible simple, undirected graphs, including cyclic and disconnected graphs, is:*

$$|\{G\}| = 2^k \text{ where } k = \frac{1}{2}n(n-1) \tag{4.14}$$

Proof: Consider the n-by-n boolean connectivity matrix N where element $N_{i,j}$ is either 0 or 1 to indicate a connection or lack of a connection between nodes i and j. We ignore elements of the matrix diagonal: $N_{i,j}$ $\forall i = j$ as nodes implicitly connect to themselves. This leaves $n^2 - n$ matrix elements. Since the graph is undirected, a connection between i and j need only be represented by $N_{i,j} = 1$ or $N_{j,i} = 1$, but not both. Thus, we require $k = \frac{1}{2}n(n-1)$ elements of the connectivity matrix N to represent all possible graphs. Note that this is simply the arithmetic series. Each element of the matrix is either "on" or "off" to represent a connection or lack of a connection. Thus there are 2^k different simple, undirected graph permutations allowing for cyclic and disconnected graphs. ▮

Instead, we will use local search techniques to approximate the optimal graphs in Section 5.3.2 of the evaluation chapter. The simulator populates each node with actual files and queries captured from a portion of the live Gnutella network. We ignore any structure while capturing Gnutella data; we are interested only in the files and queries. The simulator randomly connects the network so that each node has the same degree (k-regular). Simulation proceeds in rounds that alternate between adding a random edge and removing a random edge. After each "move," the network utility is computed.

Network utility is determined by taking the queries of each node and propagating them to all neighbors within depth t where t is the TTL. If the utility of the network decreases as the result of the last move (i.e. the randomly added or removed edge hurts global utility), the move is undone. In this fashion, the utility is monotonically increasing. To avoid local minima, the algorithm restarts with a new randomly connected network after p rounds with no increase in utility. In practice for networks of the size we consider here, a value of $p = 5000$ empirically works well.

4.11 The Cost of Being Selfish

In this subsection we provide intuition behind the utility difference between the globally optimal graph and the selfishly constructed graph. We find four main interrelated causes of non-optimality: anarchy, indifference, myopia and ordering.

4.11.1 Anarchy

Recall that some individual nodes may be unhappy in order to maximize global welfare. The ratio between the socially optimal utility and the selfish utility is known as the "Price of Anarchy".

Consider the three nodes with their files and queries as shown in Figure 4-11. Assume queries propagate two hops, i.e. $TTL = 2$. Both nodes N_0 and N_1 wish to communicate with N_2. But neither node is offering files. Node N_2 derives no benefit from the connection. Thus in a purely selfish environment the network will be disconnected.

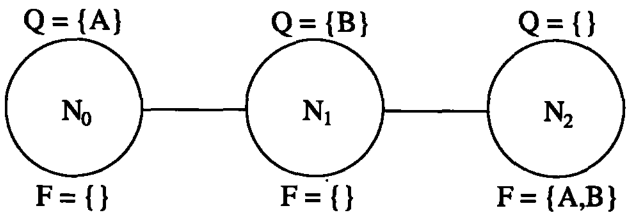

Figure 4-11: Example Network Illustrating the Price of Anarchy

From the perspective of maximizing the global utility however, a connected graph is optimal. As long as the impact of the ostracized nodes is less than the benefit according to our utility function, the optimal algorithm will connect the nodes.

Assume that nodes N_0 and N_1 send queries at a rate of one every ten seconds such that $L_0 = L_1 = 0.1$. For the utility function defined previously in Equation 4.13, the utility of the network as pictured in Figure 4-11 is $\sum_i u_i = 2(1 - 0.2\alpha)$. Solving α for $\sum_i u_i > 0$ easily confirms the simple intuition that for all $\alpha < 5$, the utility of connecting is globally better than the disconnected network of singletons resulting from selfish construction. For example, $\alpha = 1$ yields a global utility of 1.6 when connected as in the figure. In contrast the selfishly constructed network has a utility of 0.

This intuition generalizes to other networks, and nodes that are not solely free riders. Depending on the utility function, similar results are obtained for arbitrary TTL and content.

4.11.2 Indifference

The second phenomenon which leads to globally sub-optimal networks is due to indifference. We term this the "Price of Indifference." Consider two nodes with files and queries as depicted in Figure 4-12. Again, assume $TTL = 2$. Assume that each node sends its queries every ten seconds. The query load from the nodes are $L_0 = 0.1$, $L_1 = 0.2$.

Let $\alpha = 5$. If node N_0 is connected to N_1 $u_0 = \sqrt{1} - \alpha(0.2) = 0$. Even though N_0 gains utility from the successful "A" query, it loses the same amount of utility from the load induced by connecting to N_1. Thus N_0 is utility *indifferent* to N_1.

The disconnected network naturally has a utility of 0. However, the individual utilities of N_0 and N_1 interconnecting are $u_0 = 0$, $u_1 = \sqrt{1} - \alpha(0.1) = 0.5$ for a global utility of $u = 0.5$ in this example.

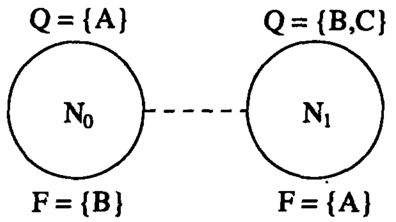

Figure 4-12: Example Network Illustrating the Price of Indifference

4.11.3 Myopia

Finally, we introduce the notion of "Price of Myopia" which is closely related to the issues of indifference. Nodes have incomplete information and only make decisions based on their current set of neighbors. We assume that given enough time to explore, nodes can find other nodes which are best able to satisfy their queries. Instead we are interested in the more subtle effect of myopia where nodes are blind to moves and topologies to which they are indifferent, but would increase global utility.

To illustrate myopic behavior, we give an example in Figure 4-13. We consider two different worlds in this figure. In world one, node N_0 and N_2 are connected while N_1 is disconnected. In world two, nodes N_0 and N_2 both connect to N_1. The following three properties hold in this example:

1. The utility of node N_0 in both worlds is the same: $U_{world1}(0) = U_{world2}(0)$. N_0 is utility indifferent to either world.

2. All nodes in each world are in equilibrium: $U_{world1,2}(N_i) \geq 0 \ \forall i$ and $\nexists$ coalition $c \in nbrs(N_i)$ such that $U_{with\ c}(N_i) > U(N_i)$.

3. The global utility in world two is greater than in world one: $\sum_i U_{world2}(N_i) > \sum_i U_{world1}(N_i)$.

Assume that each node sends its set of queries every τ seconds. Let $\varphi = \frac{\alpha}{\tau}$ to simplify the discussion. For the three nodes given, condition 1, $U_{world1}(N_0) = U_{world2}(N_0)$, implies: $\sqrt{1} - \varphi = \sqrt{2} - 3\varphi$. Thus, $\varphi \simeq 0.207$. Note that in both worlds, all nodes have non-negative utility and nodes have no incentive to deviate from their current peers. For all nodes, no coalition exists that is a subset of the current neighbor set and gives a higher utility to that node.

$\sum_i U_{world1}(N_i) = 2 - 3\varphi$ and $\sum_i U_{world2}(N_i) = 2\sqrt{2} + 1 - 10\varphi$. Thus, for $\varphi < 0.261$, the utility of world two is higher than in world one. In particular, for $\varphi = 0.207$, $U_{world1} = 1.379$ and $U_{world2} = 1.758$. Thus, the Price of Myopia in this example is $\frac{1.758}{1.379} = 1.275$.

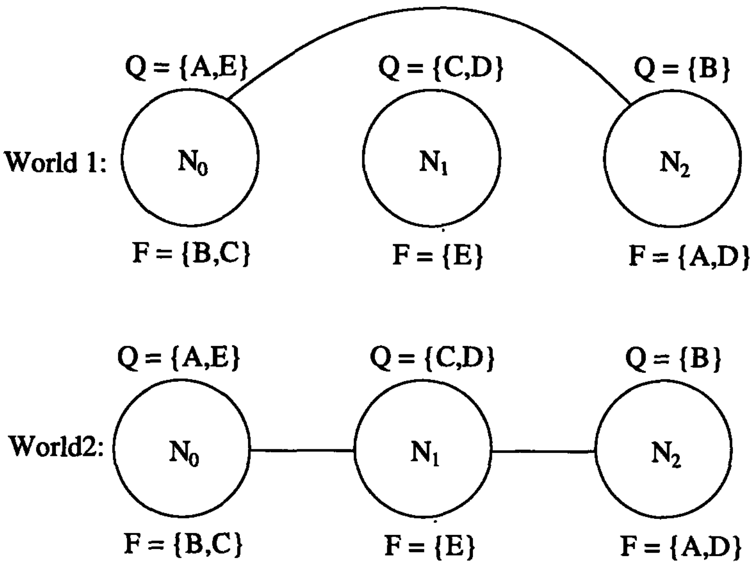

Figure 4-13: Example Network Illustrating the Price of Myopia

If selfish nodes try to minimize their connectivity by dropping connections to which they are indifferent, N_0 and N_2 will connect to each other instead of using N_1 as *transit*. N_1 will be disconnected. Note that this outcome does not contradict the behavior of utility maximizing nodes.

4.11.4 Ordering

In this subsection, we demonstrate that the ordering of events, such as exploration and decision making, affects the eventual outcome of the system.

Consider the network of four nodes as show in Figure 4-14. Node N_0 has two different coalitions that satisfy her queries. The first coalition is to simply connect to N_1. The second coalition includes N_2 and N_3. In either coalition, the derived benefit is the same (both queries are satisfied). However, N_1 prefers the coalition including connections to N_2 and N_3 because they collectively induce less load than N_1 alone.

If N_0 and N_1 connect, and then N_0 explores N_2, N_0 will maintain its connection to N_1 since N_1 provides answers to all its queries and N_2 provides no additional benefit. Since N_2 can only answer one query, N_0 will not drop N_1 in lieu of N_2. Similarly, if N_0 explores and finds N_3 while connected to N_1, N_0 will maintain its connection to N_1.

Therefore we clearly see that ordering of events and decisions affects the global and individual utilities in our system. This source of non-determinism in turn also affects our ability to realize optimality.

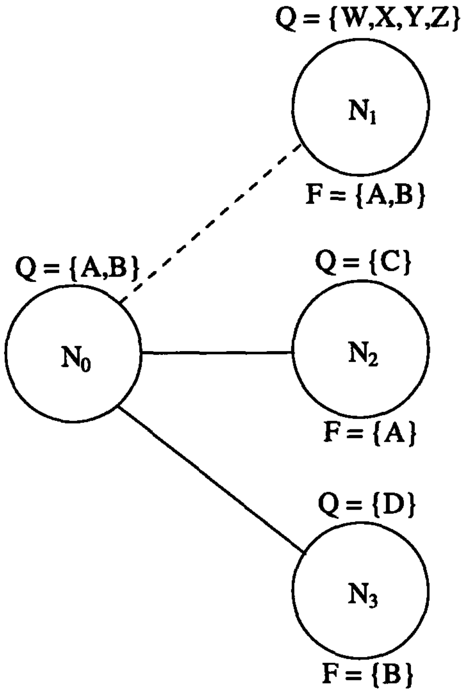

Figure 4-14: Example Network Illustrating Problems in Ordering Decisions

4.12 Summary

In this Chapter we rigorously defined ideal and fair networks as well as providing a continuum of optimality measures. We examined existing structured and unstructured overlay designs and their applicability to our design goals. Based on our analysis, we chose to build our system based on unstructured systems.

We examined three possible architectures: the current Gnutella-like unstructured architecture, an augmented design with leaf nodes that reorganize around the SuperPeers and a homogenous network of nodes that self-organize in a distributed fashion without relying on the altruism of any players in the network, in particular the SuperPeers.

We looked at the importance of Bloom filters in the existing architecture and analyzed the false positive rate as derived from our collected data. From simulation runs, we found, not unexpectedly, that the current architecture is highly dependent on the filtering capability and the willingness of the SuperPeer network to absorb the majority of the query burden.

As a possible architecture, we performed basic simulations where leaf nodes reorganize in a SuperPeer network. From our simulations we made a number of observations including the fact that the SuperPeers can limit the achievable global utility in particular circumstances. We were able to obtain much higher utility in a contrived example network, validating initial assumptions of our design.

We then tackled the crux of our thesis, a world in which SuperPeers no longer exist or are untenable for whatever reason. We described several problems and rationale for our design decisions. For example, in response to uncooperative nodes in the system (bullies), we used a hysteresis mechanism based on simulated annealing. We also found that actions cannot be made in isolation, but rather amongst coalitions of the existing neighbor set. Our design included a discussion on the selection of the utility function and its shape and discount factor α.

Next, we looked at the problem of finding optimal graphs and proved the fact that enumerating the entire set of possible graphs is exponentially hard. Our desire to find the optimal network is motivated by wanting to compare our resultant, reorganized topologies with optimal topologies. We use our results in this Chapter to design an optimal topology search strategy in our next Chapter.

Finally, we examined reasons why our distributed selfish algorithms may not perform as well as the optimal topologies. We introduced four reasons for utility suboptimal self-reorganizing networks: anarchy (selfish behavior), indifference, myopia and ordering to give further basis for our subsequent analysis and results.

The results were more often surprising than successful, but he felt it was worth it for the sake of the few occasions when it was both.

- Douglas Adams

Chapter 5

Evaluation

The previous Chapters have presented the motivation and design of an interest-based reorganization algorithm. In addition, we have provided results from first-order simulations and analyses as supporting evidence for design decisions.

This Chapter evaluates the performance of our algorithm against hypothetical optimal networks, those created by a central planner with complete information. More importantly, we evaluate the reorganized networks against the existing architecture. Our conjecture has been that the algorithm provides a better realization of fairness and optimality within a region of P2P nodes. Even more drastically, we may imagine a world where the currently presumed altruism of a large class of nodes in the system (the SuperPeers) disappears or is no longer applicable. Intuitively it is clear that nodes will realize a higher degree of fairness as they act selfishly and thus by definition will take actions that are individually beneficial. The downside to providing fairness and allowing nodes to act autonomously however is the potential for loss of aggregate utility, or global welfare. In Chapter 4 we provided several reasons for the utility differences seen in a reorganized versus an optimal network.

We begin this Chapter by revisiting the existing heterogenous architecture of SuperPeers and leaves. Using our methods of simulation described in previous Chapters and detailed further here, we evaluate the individual and global utility of this base model. All of our simulations are driven by the real Gnutella data captured from a live portion of the network and described in Chapter 3. To provide a metric of utility equilibrium in the base model, we measure the utilities in a heterogenous network under the assumption of individually rational leaves. Thus our first two sections provide metrics of the existing architecture we use later to compare performance of our algorithm with.

Next we use search techniques to find solutions for the three types of optimal networks we described in Chapter 4: social, risk-adverse altruistic and equilibrium optimal. Comparison and discussion of the optimal networks highlights several problems involved in a purely selfish reorganization algorithm. The most detrimental of which is the simple bartering problem, causing a large fraction of nodes to fully disconnect because there exists no suitable peer node. Analysis of the degree structure of the resulting optimal topologies provides further insight into how our algorithm behaves on real data and nodes.

We then simulate our interest-based reorganization algorithm and show that it achieves a global utility very close to the predicted equilibrium optimal utility while disconnecting free-riders. This result is encouraging, however the global utility falls far short of the utility obtained in the base and individually rational SuperPeer models.

Matching the utility of the existing architecture provides the impetus for the subsequent section: an analysis of the addition of Bloom filtering to all nodes in the system. We model a hypothetical reorganization model where the utility imbalance is nullified by blocking extraneous queries, i.e. those which cannot possibly elicit an answer. Using Bloom filters we show that the global utility is within that realized by the SuperPeer models.

Extensive supporting data from the simulations is provided for each situation including global utility as a function of reorganization round, the difference in individual node utility between different networks, message traffic in the system and success rate. Each of these metrics provides additional insight into the performance of our reorganization algorithm.

We perform a third type of simulation using a technique borrowed from the game theory literature: epsilon equilibrium. Epsilon equilibrium relies on the premise that it is impossible to accurately predict the utility function for every player in the system, therefore nodes continue to make selfish decisions but expand the strategy space to include all moves within ϵ of the selfish move. We find that by employing such a model, the disconnection problems of our other simulations disappear and our reorganization algorithm finds a solution where over 60% of the nodes are connected. Most encouraging is that this model converges to a utility 34% greater than achieved in the SuperPeer models while making no assumptions on the benevolence of nodes or centralized organization.

Finally, this evaluation Chapter concludes by examining the community structure of the resultant networks. We employ several recent algorithms to quantitatively measure the inherent structure of our topologies including an analysis of the connected components, betweenness centrality and modularity.

5.1 Random Graph Construction

Before presenting the evaluation of our algorithm, we briefly diverge to discuss a simulation particular. Recall that our captured traces ignore topology. We are interested only in the content and queries with no presumptions of the topology. Thus, to accurately simulate the network structure, we must construct random graphs with particular properties. While this Section shows some of the non-trival issues involved, the casual reader may skip to the next Section.

When randomly constructing graphs of interconnected nodes representative of the Gnutella architecture our goal is to create networks where every node is connected to d other nodes at random. d may be drawn from a probability distribution or may be constant. When d is constant, we say that the graph is *d-regular*. In logical networks there is no reason to have more than one undirected edge between any pair of nodes; such graphs are known as *simple*. We place no restrictions on whether the graph is *connected*, that is whether a path exists from every node $i \in V$ to every other node $j \in V$ in graph $G = (V, E)$. In fact our algorithm may encourage disconnected islands of nodes with similar interests. Finally, the graph is permitted to contain *cycles* where a subset of G's edge set E forms a path such that the first and last nodes are the same. In this section we detail our method for initially connecting our simulated networks.

In particular, we want to generate simple d-regular graphs, possibly disconnected or containing cycles, as though they were drawn from the uniform distribution of all possible d-regular graphs. It is not possible to generate a d-regular graph for all numbers of nodes. For d even, and n nodes, any $n > d$ may be interconnected to generate a d-regular graph. For d odd, only even n where $n > d$ may be used to generate the d-regular graph. Using the following theorem, we only generate initial connectivity graphs for which d-regular constructions exist.

Lemma 5.1.1 *The number of nodes N which produce a d-regular graph G is lower-bounded by* $d + 1$.

This is trivially true since each node n_i must connect to at least d other unique nodes.

Lemma 5.1.2 *The number of nodes N which produce a d-regular graph G must satisfy the condition* $2|dN$ *(the product of d and N is even).*

Each edge e connects a pair of nodes. Let the degree of node i be $d(i)$. Placing an edge between nodes i and j requires that $d > d(i)$ and $d > d(j)$. The total number of possible pairings is dN. If dN is not even then there is no possible way to place edges between nodes such that $d(i) = d, \forall i \in G$. Given this intuition and the previous two lemmas, we have the following theorem.

Theorem 5.1.3 *The number of nodes N which produce d-regular a graph G is given by:*

$$N = \begin{cases} i > d \, \forall i & \textit{if } d \textit{ mod } 2 = 0 \\ i > d \, \forall (i \textit{ mod } 2 = 0) & \textit{if } d \textit{ mod } 2 \neq 0 \end{cases} \tag{5.1}$$

5.1.1 Fast Random Regular Graph Generation

One method for choosing a d-regular graph at random is to generate the entire set of d-regular graphs for a given number of nodes n and degree d. Draw a graph uniformly at random from this set to output a d-regular graph. However, as shown previously in Theorem 4.10.1, this is an exponentially large set and thus intractable to generate. Instead we use the algorithm of Steger and Wormald [46]. Their algorithm, we which term $GEN(n, d)$ on n nodes to create simple d-regular graphs, runs in expected time $O(nd^2)O(1)$ for $d = O(\sqrt{n})$. $GEN(n, d)$ can be summarized as the following four-step process:

1. Create nd unpaired points: $U = \{1, 2, ..., nd\}$. These represent n groups of d points.

2. Until no suitable pairs remain: choose i, j randomly from U. If i, j are *suitable* (defined below), connect them and remove i, j from U.

3. For all pairs i, j, create graph G such that nodes i mod n and j mod n have an edge between them.

4. Test: if G is d-regular, output it, otherwise retry.

Where a "suitable" pair i, j is defined as one where $i \neq j$ and $group(i) \neq group(j)$. Figure 5-1 shows the partial execution step 2 of the Wormald algorithm graphically for a $n = 6$ $d = 3$ network construction. Each column represents a group for a single node. Connection a (the line drawn in the Figure with label "a") denotes a link between node 1 and node 2. It is suitable since there are no other links between group 1 and 2. b similarly connects node 2 to 4. However, connection c is not suitable since there is already connection b between groups 2 and 4.

We give pseudo code of our implementation of the Steger-Wormald algorithm using a single dimensional array in Algorithm 5.1. The algorithm constructs array P consisting of the nd points

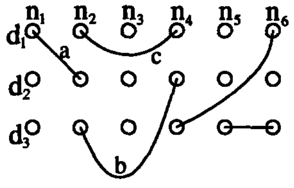

Figure 5-1: Steger-Wormald Pairwise Connection of nd Vertices

to be paired. While there are pairs remaining to connect, the algorithm gets a random pair and determines if they are suitable via Algorithm 5.3. If the two points can be connected, the pair array is reflected such that position in the array reflects the vertex number to which it connects.

After a creating suitable pairwise connections for all nd points, the algorithm creates a graph G. For every point $i \in P$, determine the corresponding group $j = i \bmod n$. Add an edge between node j and the value that position i in array P points to: $e_{j,P[i]}$. Add this edge to G and output the final graph.

```
for $i = 0$ to $nd - 1$ do
  $P[i] \leftarrow -1$
$pairs \leftarrow \frac{nd}{2}$
while $pairs > 0$ do
  $suitable$ = false
  while $notsuitable$ do
    $i, j \leftarrow$ GetRandPair()
    $suitable$ = Suitable($i, j$)
  $P[i] = j$
  $P[j] = i$
  $pairs \leftarrow pairs - 1$
for $i = 0$ to $nd - 1$ do
  $j = i \bmod n$
  $E \leftarrow E + e_{j,P[i]}$
return $G = (V, E)$
```

Algorithm 5.1: $GEN(n, d)$: Steger-Wormald d-regular Graph Generation

Algorithm 5.2 simply selects two unmatched points x, y where $x \neq y$ at random from the range 0 to $nd - 1$.

To determine if two points are suitable for connection, we simply find whether there is an existing connection between the same group. An existing connection means that an additional connection would create a parallel edge such that two points are connected by more than one edge and thus our graph would no longer be simple. Algorithm 5.3 checks each member of the group. Arithmetic is done modulo n to easily break the single dimensional array into n groups.

```
$x \leftarrow [0, nd)$
$y \leftarrow [0, nd)$
**while** $P[x] \geq 0$ **do**
  $x \leftarrow [0, nd)$
**while** $P[y] \geq 0$ or $x = y$ **do**
  $y \leftarrow [0, nd)$
return $(x, y)$
```

Algorithm 5.2: *$GetRandPair()$*: Steger-Wormald Pair Finder

```
**for** $i = 0$ to $d - 1$ **do**
  **if** $(P[x + (ni)] = y) \bmod n$ **then**
    return $false$
return $true$
```

Algorithm 5.3: $Suitable(x, y)$: Steger-Wormald Suitable Pair

Note that the algorithm may reach deadlock when unconnected pairs remain and those unconnected pairs are all in the same group. In practice, we supplement our algorithm with a check such that the algorithm restarts when there are un-connectable pairs remaining.

5.2 SuperPeer Network Simulation

The first step in evaluating our scheme is to revisit the existing heterogenous two-level hierarchy which we previously termed the "Base Model." Section 4.6.1 showed the number of messages in the system, the result of a first-order simulation. Using the utility function of Section 4.6.2, we extend the simulation and analysis to include modeling the individual node and global utilities. These utilities, in conjunction with the query success rate, provide a baseline against which to evaluate our proposed reorganization algorithm. We then consider similar metrics for a SuperPeer network with individually rational leaves.

Our objective in this Section and the next is to show the current level of utility and happiness achieved with the existing architecture. We then systematically tear down the implicit assumptions of altruism built into the existing architecture while showing the resulting negative impact on utility. Once we have a selfish equilibrium, presumably with much lower global utility, we show the ability of our algorithm to reorganize and restore the utility of individual nodes and the system as a whole to similar levels as realized in the SuperPeer network.

5.2.1 Base Model

We begin our analysis by examining the performance and global utility of today's SuperPeer networks. Gnutella specifies a number of default network properties we maintain as previously detailed in Table 4.1. SuperPeers are randomly connected to form a d-regular graph using the method described in detail in Section 5.1.1. Each leaf is randomly connected to C_s different SuperPeers, subject to the constraint that a SuperPeer accepts at most C_a leaves. To reiterate, a run of the system in our simulator is the following four step process:

1. Each node sends all of its queries.
2. According to the TTL, the queries are flooded through the SuperPeer network.
3. According to the TTL and SuperPeer Bloom filtering, queries are propagated to the leaves.
4. Total query traffic and the number of successful query matches are recorded.

We establish the query success baseline by computing ideality (Section 4.1.1), the maximum possible number of successful queries if every leaf node were directly connected to every other leaf. Recall that P_i is the set containing all possibly answerable queries of i. Then the query success baseline is: $P_{max} = \sum_{i \in N} |P_i|$.

We simulate the base model network to determine the TTL that provides the highest utility. In the SuperPeer network, we are concerned only with the global utility given that the SuperPeers perform bloom filtering to eliminate unnecessary traffic downstream toward the leaves. Table 5.1 summarizes the simulation results; we list the utility without bloom filtering for comparison only. As the TTL increases, the "reach" of the queries improves increasing the success rate. At a TTL of 5, the network provides responses for 96.0% of P_{max}. However, in spite of the bloom filters, utility drops past a TTL of 4. Thus, there is an inflection point in the utility obtained versus TTL with a maximum utility of 213. The utility drop is due to multiple query requests arriving at the SuperPeers from different ingress connections. Because the SuperPeers do not maintain state on which queries have already been forwarded on to their leaves, the leaves receive multiple copies of the same query. [1]

[1] This is a modeling assumption rather than an architectural issue and may not necessarily be true in all implementations. However, it does not quantitatively affect our end results.

Table 5.1: Network Performance in Base Model Consisting of Two-Level Heterogenous Network of SuperPeers and Leaves

TTL	Utility (No Filtering)	Utility	% Success
2	-1421	64	0.082
3	-8943	198	0.399
4	-46089	213	0.648
5	-233515	-289	0.960

Table 5.2: Network Performance in Individually Rational Model Consisting of Two-Level Heterogenous Network of SuperPeers and Individually Rational Leaves

TTL	Utility (No Filtering)	Utility	% Success
2	2	2	0.002
3	21	27	0.102
4	-214	92	0.192
5	-1328	130	0.308

5.2.2 Individually Rational Model

Next we examine the same network topology and participants (same graph G as in the base model), but in a context where the leaves are individually rational, a notion we introduced in Section 4.6.2. The action space is whether a leaf maintains its set of SuperPeer connections or disconnects one or more of them. That is, given our utility function, a leaf will choose to disconnect members of its SuperPeer set if doing so will improve the node's utility.

We run our four step simulation again. Table 5.2 displays the global utility and success rate in the SuperPeer network with individually rational leaves. Note that the success rates are much lower and the utility is only 61% of that achieved in the original network. Because individually rational nodes may disconnect, the success rate is only 31%.

The utility of each node is plotted in Figure 5-2 for both the base model and the individually rational model. In the case of individual rationality, the utility of many nodes is zero as they fully disconnect. This implies that in the current architecture, a majority of nodes have an incentive to deviate from their SuperPeers, suggesting a possible reason for the lack of stability seen in our measurement data.

Since the individually rational model is representative of an equilibrium in the current architecture, we take the maximum utility obtained, 130, as the network utility achieved in a Gnutella-like

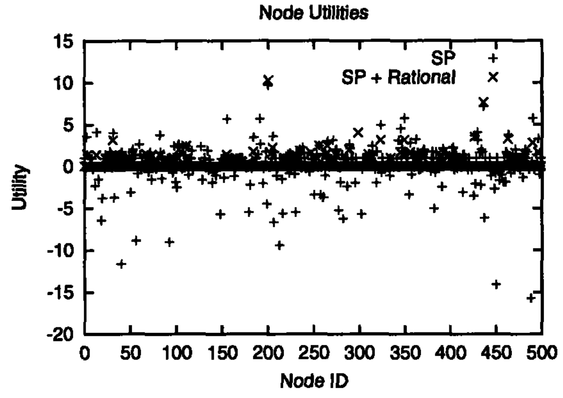

Figure 5-2: Node Utility in SuperPeer Network with Individual Rationality

system. In the next Section, we proceed to examine our full model including a homogenous network with reorganization.

5.3 Homogenous Network Simulation

Having established the performance of our data set on the base and individually rational SuperPeer topologies and mechanisms, we next seek to understand the performance of our reorganization algorithm. However, before simulating and measuring the performance of the algorithm, we first determine optimality for the network without SuperPeers. In Chapter 4 we defined three types of optimality: social, risk-adverse altruistic and equilibrium. This Section provides simulation results that find each type of optimality. Understanding the optimal, ideal network that maximizes global utility provides a basis with which to compare the selfish algorithm.

As proved in Section 4.10, it is exponentially hard to enumerate and test all possible graph structures for the purpose of finding an optimal one. Instead, we use search techniques to approximate the optimal graph. We start with a fully disconnected graph. With no connections, every node has a utility of zero and is thus in equilibrium. Simulation proceeds in rounds that include adding a random edge and removing a random edge. After each "move," the network utility is computed. If the network utility is greater than any previous utility, and the optimality conditions are met, the move is kept. Otherwise the move is reversed. The optimality conditions differ according to their definitions. To avoid local minimums, we restart after p moves without any utility improvement. In practice for networks of the size we consider here, a value of $p = 5000$ empirically works well. In our simulations, we restart a total of $k = 10$ times.

5.3.1 Risk-Adverse Altruistic Optimal

Consider Risk-Adverse Altruistic optimality, as defined in Section 4.2.2, where all nodes are "happy," i.e. their individual utilities are non-negative. During the simulation, if the utility of each node after the move remains non-negative and the global utility improves, the move is kept. Otherwise the move is undone. More formally, the search for Risk-Adverse Altruistic optimality can be summarized in the pseudo code of Algorithm 5.4.

```
for $i = 1$ to $k$ do
  Disconnect graph $G = (V, E)$ s.t. $E = \{\}$.
  $no\_improve \leftarrow 0$
  while $no\_improve < p$ do
    Add random edge $e_1 \notin E$
    $score \leftarrow util(G)$
    if $U_i \geq 0 \; \forall i \in V$ and $score > best\_score$ then
      $no\_improve \leftarrow 0$
      $best\_score \leftarrow score$
    else
      remove $e_1$
      $no\_improve \leftarrow no\_improve + 1$
    Delete random edge $e_2 \in E$
    $score \leftarrow util(G)$
    if $U_i \geq 0 \; \forall i \in V$ and $score > best\_score$ then
      $no\_improve \leftarrow 0$
      $best\_score \leftarrow score$
    else
      add $e_2$
      $no\_improve \leftarrow no\_improve + 1$
```

Algorithm 5.4: $raoSearch(G)$: Search for Risk-Adverse Altruistic Optimal

Figure 5-3 displays the search for risk-adverse altruistic optimal utility. We find the best utility of 200, approximately the same as achieved in the SuperPeer network and a similar success rate of 54%. In both optimal non-SuperPeer network and the individually rational SuperPeer network, all nodes have non-negative utility, but the optimum utility found in the non-SuperPeer network is significantly better.

5.3.2 Equilibrium Optimal

However, the optimality we are most interested in is an equilibrium optimality, as defined in Section 4.2.3, where nodes have both non-negative utility and no way of dropping a connection to improve their utility. In other words, a selfish node has no incentive to drop a link to further increase her util-

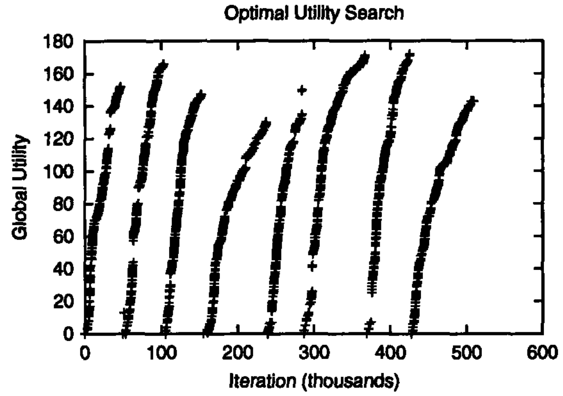

Figure 5-3: Search for Risk-Adverse Altruistic Optimal Utility

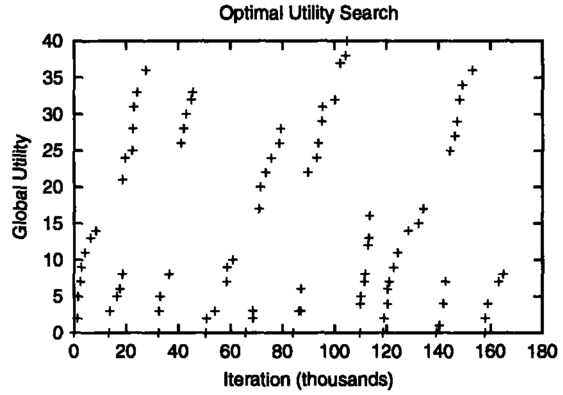

Figure 5-4: Search for Optimal Equilibrium Utility

ity at the expense of the global utility. Figure 5-4 displays the search for optimal global equilibrium utility. The utility is much lower than in the search for risk-adverse altruistic optimum, with utilities between 30 and 40 and success rates from 10 to 15%. Thus we see that the utility of an optimal network with purely selfish nodes is severely limited.

The algorithm for finding equilibrium optimal is essentially the same as Algorithm 5.4, with the exception of the edge add or remove accept condition. To be equilibrium optimal, each node must not only have non-negative utility, it must also have no incentive to deviate from its current set of neighbors. Thus, if any node can remove an edge to selfishly increase its individual utility, the move is reversed. When no nodes have any desire to change their current set of peers, the network is in equilibrium.

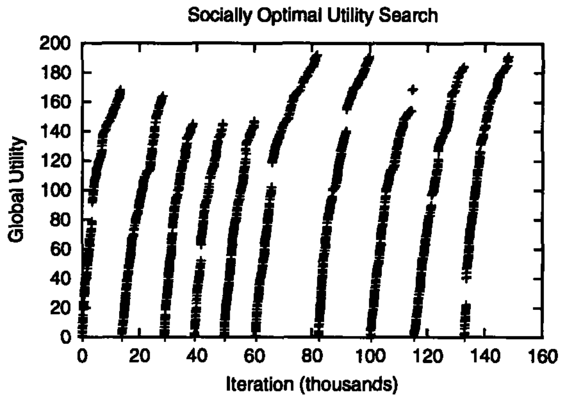

Figure 5-5: Search for Socially Optimal Utility

5.3.3 Socially Optimal

Finally, for completeness, we search for the socially optimal network. A socially optimal network is not in a stable equilibrium for the individual players, but provides further intuition about the organization of the network. Again, the algorithm is essentially the same as Algorithm 5.4, except that the only condition for accepting a move is whether or not it increases the global utility. Thus our simulator plays the part of the oracle coordinating connection decisions to maximize aggregate utility.

Figure 5-5 displays the search for the socially optimal utility. Surprisingly, the socially optimal utility achieved is almost identical to that seen in the risk-adverse altruistic optimum search, around 200. We attribute this to the fact that a purely selfish node disconnects peers unless a close balance exists. In contrast, the risk-adverse altruistic model assumes that nodes are willing to accept a slightly lower payoff, so long as they have non-negative utility, in return for higher global utility. The imbalance and its effects are discussed later in the Chapter when we present more details on the bartering problem. The similarity between utilities in the social and risk-adverse altruistic optimum provide motivation for our later examination of epsilon equilibrium in Section 5.4.

Table 5.3 summarizes the best utilities found from each of our three types of optimality using our search technique. One important point contained in this summary is that in an environment with a centralized point of control and complete knowledge, it is possible to construct a network topology with utility and success rates comparable to those found in the SuperPeer models. In addition, the optimal networks achieve these utilities with only a TTL of 2, resulting in much less network traffic. However, we use the optimal networks as a basis for comparison as we are primarily interested in

Table 5.3: Best Utilities and Resulting Success Rates Found for Different Optimality Types Using Complete Knowledge Search with Restart

Optimality Model	Description	Utility	% Success
Social	Maximize global welfare	192	0.473
Risk-adverse altruistic	Nodes maintain non-negative utility, sacrifice selfish maximum for global welfare	172	0.463
Equilibrium	Purely selfish behavior, nodes have no incentive to deviate from neighbor coalition	40	0.170

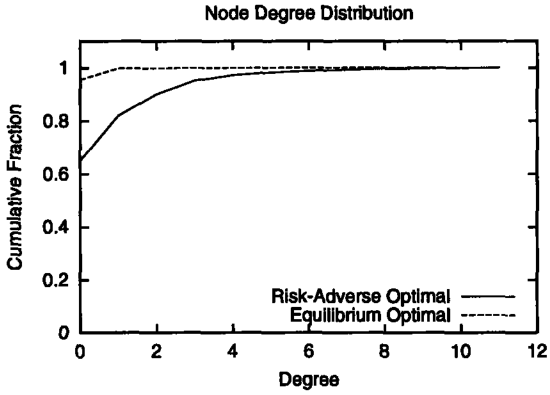

Figure 5-6: Node Degree Distributions of Optimal Networks

the design of a selfish distributed mechanism.

5.3.4 Optimal Network Structure

We next consider the basic structure of the optimal networks. Figure 5-6 displays the node degree distribution for both the risk-adverse altruistic and equilibrium optimal networks. In the risk-adverse altruistic optimal network, approximately 65% of the nodes are completely disconnected and have degree 0. For the equilibrium optimal network, fully 95.6% of the nodes are disconnected. The large number of disconnected nodes is due to the bartering problem.

Figure 5-7 provides a simple example of the bartering problem. Node N_1 sends a query for a to N_2. Since N_2 has a, this yields positive utility u_a to N_1 and the system. Similarly, N_2 sends a query for z to N_1. Since N_1 does not hold z, and the query imposes a non-negligible load, N_1 cannot be part of N_2's best coalition. Therefore N_2 will disconnect N_1. Say that the reduction in utility seen by N_2 due to maintaining a coalition that includes N_1 is u_z. The nodes will not remain connected, even if $u_a \gg u_z$. This phenomena is detrimental to the final system utilities we see in all rational

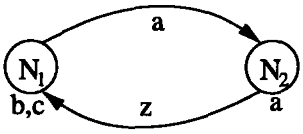

Figure 5-7: Bartering Between P2P Nodes

networks. We return to this problem and present a modeling-based solution in Section 5.4.

5.3.5 Reorganization with the Full Model

We simulate our reorganization to determine the difference in utility between the distributed algorithm, as described by the full model in Section 4.9, optimal and the base model.

During reorganization, nodes must make several decision per round, whether to:

1. Connect to another node (explore)

2. Allow other nodes to connect (passive explore)

3. Drop connections to nodes in order to improve utility (determining the best coalition)

In order to overcome the bully problems and ordering problems described in Sections 4.9.2 and 4.11.4, we employ simulated annealing, a popular technique for large optimization problems as described in Section 4.9.3. In simulated annealing, there is some probability that the non-selfish action is taken in order to escape local minimums. Initially the probability of taking a non-optimal action is high and over time it approaches zero. The probability time decay is referred to as "cooling" and is typically an exponential function.

Figure 5-8 shows the global utility during reorganization. We see that the utility quickly switches from negative to positive as nodes drop harmful connections in their coalitions. As nodes explore and become progressively happier, the system converges toward the previously derived optimum, between 30 and 40. Thus, we see that in a relatively short number of rounds, the network selfishly reorganizes to achieve a global utility comparable to the optimal equilibrium. In addition, the query success rate is approximately 22%, well within the range established by the individually rational model.

Finally, we examine each individual node's utility change between being in the SuperPeer network and in the reorganized bloom-filter homogenous network. Figure 5-9 plots each node in a plane with the two situations on each axis to illustrate the utility differences. The large number of

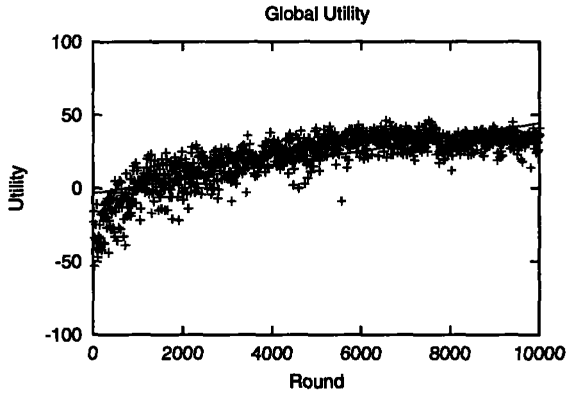

Figure 5-8: Global Utility vs. Reorganization Round

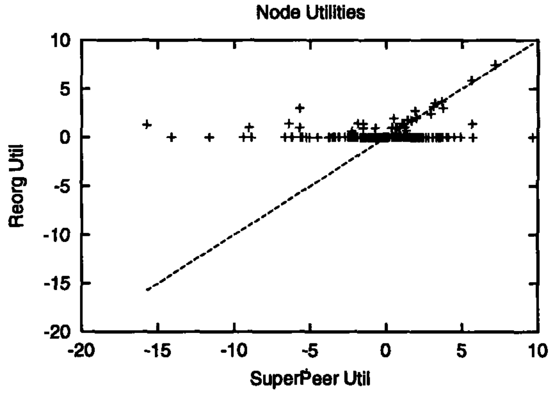

Figure 5-9: Individual Node Utilities Compared, Base SuperPeer Network vs. Selfish Reorganization

points along the utility equals zero horizontal axis are indicative of nodes that are now disconnected. Nodes along the $y = x$ line are nodes whose utility has not changed. And nodes to the left of the $y = x$ line are those whose individual utilities increased.

However, a tenable reorganization model must achieve comparable utility to that seen by the existing architecture. The major obstacle to achieving higher utilities in purely selfish networks is due to the slight imbalance of useless queries causing otherwise global utility contributors to be disconnected. Based on this observation, in the next subsection, we consider an architecture that includes bloom filters between the leaves.

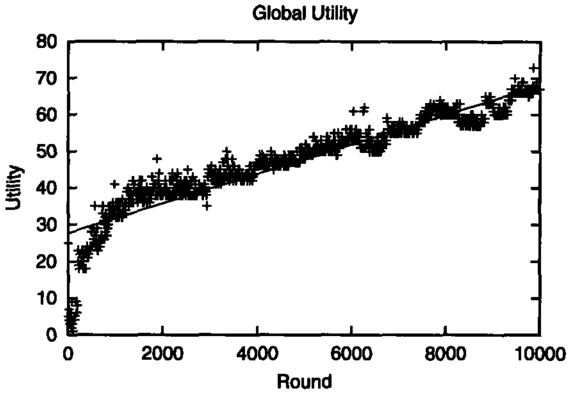

Figure 5-10: Global Utility vs. Reorganization Round with Bloom Filtering

5.3.6 Bloom-Based Reorganization

We next consider the case where leaf nodes have the same filtering ability as the SuperPeers. That is, we again assume an architecture without SuperPeers, but where each node has a bloom filter digest of the files of his neighbor.[2] In the case of queries with TTL=2, the digest is the bit-wise OR of the neighbor's file digest and the digests of the neighbor's neighbors.

Our simulation proceeds as before, collecting the node utilities and query successes. The global utility during reorganization for a network with such bloom filtering is shown in Figure 5-10.

We see that the utility is always non-negative as the bloom filters prevent extraneous queries from propagating in the network. The rate of increase, as depicted by the solid line, shows the utility increasing at a faster rate than in our previous experiment. The final utility converges to approximately 70, about twice as high as in the reorganized network without bloom filters.

Figure 5-11 displays the query success rate of our selfish reorganization scheme both with and without the hypothetical bloom filtering. We see that the success rate is increasing, but not monotonically. This is characteristic of the learning that is occurring during reorganization. As nodes are "punished" by being disconnected by other nodes with different interests, the query success rate goes down until these nodes are "rediscovered" and brought into interest clusters. Note that the query success rate approaches the same value as in the individually rational SuperPeer model, approximately 25%.

Next, Figure 5-12 displays the number of messages propagated through the system in each round. While the number of messages is also increasing over time, the increase is gradual and cost

[2] We don't consider the cost of communicating these filters between neighbors, but note that it is minimal in comparison to the cost of sending queries.

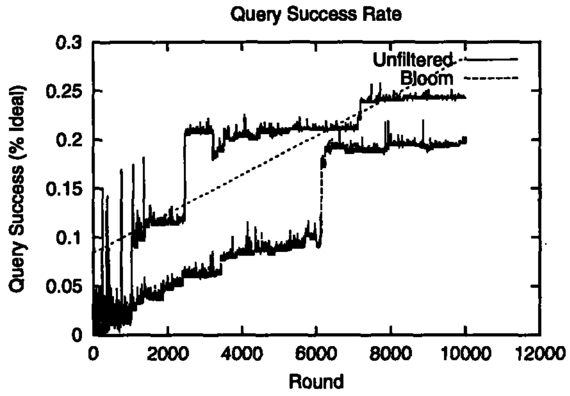

Figure 5-11: Query Success Rate vs. Reorganization Round

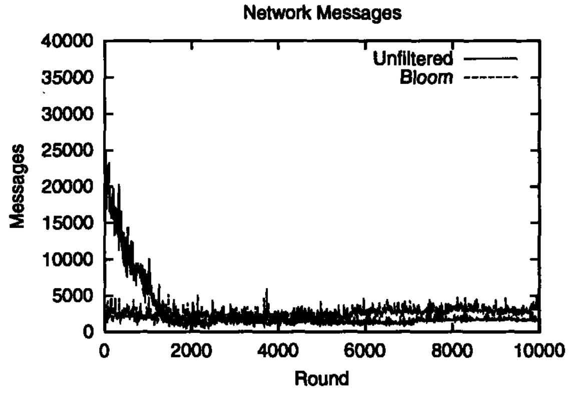

Figure 5-12: Network Messages vs. Reorganization Round

is offset by a better success rate. The number of messages in the network is approximately 1/10th that of the base SuperPeer simulation.

5.4 Epsilon Equilibrium

Careful examination of the network utility in both the search for the optimal equilibrium utility and after reorganization reveals that we are not able to match the performance obtained by the SuperPeer network. In addition, we see from Figure 5-6 that the network is very disconnected: between 65% and 95% of the nodes are singletons.

As observed by [8], game theoretical analysis of P2P systems can lead to a Nash equilibrium where nodes have no incentive to share or connect. Such a situation is known as the "tragedy of the commons," whereby nodes refuse to connect to any other node and lead the entire system to collapse. The authors suggest that mere existence of P2P networks in use operation today provide evidence that the Nash equilibrium is not reached.

In this section we introduce the notion of "Epsilon Equilibrium", a concept we borrow from the economics and game theory literature [32]. Epsilon equilibrium is based on the intuition that, for many games, it is difficult to accurately determine all players' payoff, i.e. our utility function. In order to make up for this deficiency, it has been suggested to define strategies where every player is within ϵ of the largest possible payoff. This prevents the exclusion of a potentially large set of equilibria that are more flexible and may better model reality

Using an epsilon equilibrium model, we show that the network after reorganization is dominated by disconnected nodes because nodes exhibit the bartering problem. Any two nodes must each have content the other desires, or have equivalent virtual connectivity through third party transit nodes. If not, even if one node derives significant benefit while imposing little load, the nodes will be disconnected. A purely rational node disconnects nodes that are part of a coalition that is not the best coalition.

We modify the algorithm nodes use to select their best coalition to include ϵ. For all possible coalitions, each node determines a set of coalitions that are within ϵ of the coalition with the highest possible utility. From this set, the largest coalition, i.e. the coalition including the most edge connections, is selected as the best coalition.

This simple modification to include ϵ prevents the degenerate behavior that a node that induces very little load is disconnected even when it contributes highly to the system utility.

Figure 5-13 displays the node degree distribution for various ϵ values. Empirically we see that an ϵ value around 0.1 provides good results: the network is over 60% connected.

Figure 5-14 shows the evolution of global utility using our reorganization algorithm with $\epsilon = 0.1$. The simulation reaches a utility of approximately 120 within 3000 rounds.

Finally, Figure 5-15 plots each node in a two-dimensional space with the node's individual utility in the SuperPeer network versus the node's utility in the reorganized network with epsilon equilibrium. Points above the vertical line represent nodes that see improved utility in the reorganized network over their utility in the SuperPeer network. Points below the line are nodes that experience worse utility in the reorganized network. We see that many nodes have higher utility in the ϵ-equilibrium network.

Clearly, ϵ-equilibrium illustrates the importance of the bartering problem and the ability of our reorganization algorithm to match and exceed the global utility of a SuperPeer network. In the

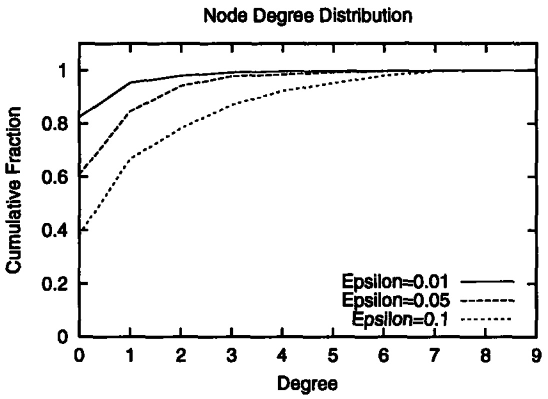

Figure 5-13: Network Degree Distribution for ϵ-Equilibrium

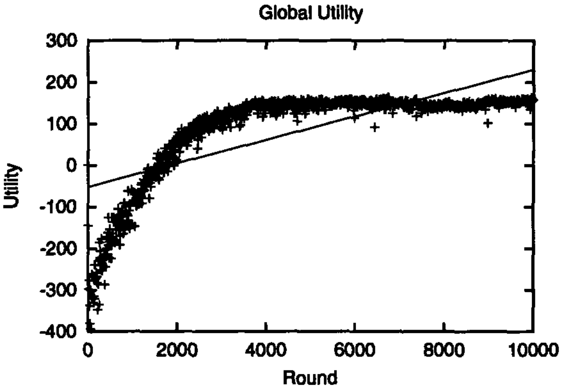

Figure 5-14: Network Utility, $\epsilon = 0.1$ Equilibrium

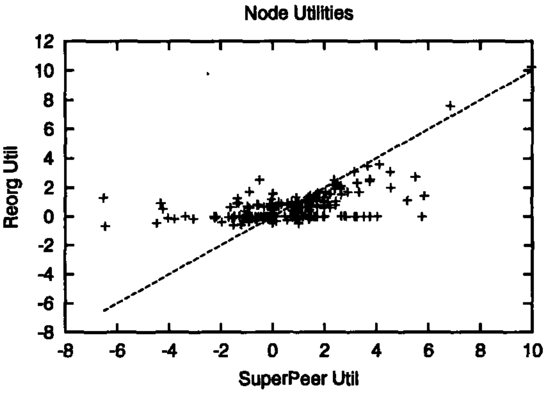

Figure 5-15: Node Utility in SuperPeer vs. $\epsilon = 0.1$ Equilibrium Networks

Table 5.4: Reorganization Simulation Utility Summary

Network Model	Description	Utility
Full	Interest-based self-reorganization; homogenous nodes	38
Full Bloom	Reorganization with Bloom Filtering	70
Full, ϵ-Equilibrium	Full reorganization modeled with epsilon-equilibrium	174

individually rational SuperPeer network we see utility of 130 and in ϵ-equilibrium we see 174, an improvement of 34%! Thus, in the ϵ-equilibrium model of node behavior, we are able to not only achieve, but exceed, the level of utility as realized in the individually rational SuperPeer network as it exists today. Table 5.4 summarizes the utilities found from our reorganization algorithm in each of our three models.

5.5 Finding Community Structure

We are interested in understanding the community structure of the overlays constructed by our reorganization algorithm. In particular, community structure provides another metric by which to measure the quantitative difference between an "optimal" overlay network versus a selfishly constructed one.

Traditional clustering algorithms employ agglomerative techniques that compute a similarity metric between vertex pairs. From an initially disconnected graph, edges are successively added in their order of similarity. Agglomerative methods are often good at identifying and connecting strongly related nodes at the core, but can neglect weaker relationships.

We use a recently developed divisive rather than agglomerative algorithm from Newman [28] that has empirically been shown to produce better results. Newman's algorithm relies on the notion of "betweenness centrality" of nodes.

5.5.1 Betweenness Centrality

For $G = (V, E)$, let σ_{st} be the number of shortest paths from s to t in G. By convention, $\sigma_{ss} = 1$. Define $\sigma_{st}(v)$ as the number of shortest paths from s to t on which $v \in V$ lies. Let the pair-dependency of nodes s, t on v be $\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$. Thus the pair-dependency is simply the ratio of the number of shortest paths between s and t that v lies on.

The betweenness centrality [16] of a vertex $v \in V$ is the ratio of the number of shortest paths involving v, to the total number of shortest paths, taken over all vertex pairs $(i, j) \in V$. In other words betweenness centrality is a metric of the *relative importance* of a particular node in the network.

Formally, the betweenness centrality for a node v is:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} = \sum_{s \neq v \neq t \in V} \delta_{st}(v) \tag{5.2}$$

In the naive approach, calculating betweenness centrality is dominated by computing the sum of pair-dependencies which has a running time complexity of $O(n^3)$. Fortunately, Brandes presents an fast algorithm that requires $O(nm)$ time and $O(n + m)$ space [6].

We modify Brandes' algorithm for vertex betweenness centrality to compute edge betweenness, i.e. the number of shortest paths traversing each edge in the network. Thus, edge betweenness can be seen as the traffic flow along an edge when all nodes in the graph source traffic to all other nodes. Formally, we define the betweenness centrality for an edge e as:

$$C_B(e) = \sum_{s \neq t \in V} \frac{\sigma_{st}(e)}{\sigma_{st}} \tag{5.3}$$

Where $\sigma_{st}(e)$ is the number of shortest paths from s to t that contain edge e.

Our edge centrality algorithm is given in Algorithm 5.5. The output of our algorithm is an adjacency matrix e where $e[i][j] = e[j][i]$ corresponds to the betweenness centrality of the undirected edge between nodes i and j in G.

The algorithm is based on a breadth-first search (BFS) traversal of the graph. The first while loop is identical to the one presented in Brandes' algorithm and simply computes the number of shortest paths between node s and all other nodes as well as the predecessor list P. $P_s[w]$ is a list of nodes that are predecessors (i.e. immediate neighbors) of w on the shortest path from s to w. In addition, the loop builds a stack S by pushing nodes onto the stack during the BFS. Thus, popping from the stack gives nodes that are of non-increasing distance from the current root node s.

The second while loop performs the accumulation of paths, beginning with the node farthest away from s. For each pair s, w, there is a corresponding predecessor list. For each predecessor $v \in P[w]$, we compute the pair-dependency and update the edge betweenness. Formal correctness guarantees are given in detail in Brandes' full paper [6].

```
$e[i][j] \leftarrow 0, i, j \in V$
for $s \in V$ do
   $S \leftarrow$ empty stack
   $P[w] \leftarrow$ empty list, $w \in V$
   $\sigma[t] \leftarrow 0, t \in V, \sigma[s] \leftarrow 1$
   $\delta[t] \leftarrow 0, t \in V$
   $d[t] \leftarrow -1, t \in V, d[s] \leftarrow 0$
   $Q \leftarrow$ empty queue
   enqueue $s \rightarrow Q$
   while $Q$ not empty do
      dequeue $v \leftarrow Q$
      push $v \rightarrow S$
      for all $w = nbrs(v)$ do
         if $d[w] < 0$ then
            enqueue $w \rightarrow Q$
            $d[w] \leftarrow d[v] + 1$
         if $d[w] = d[v] + 1$ then
            $\sigma[w] \leftarrow \sigma[w] + \sigma[v]$
            append $v \rightarrow P[w]$
   /* Finished building predecessor lists and computing #SP */
   while $S$ not empty do
      pop $w \leftarrow S$
      for all $v \in P[w]$ do
         $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]}(1 + \delta[w])$
         $e[v][w] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]}(1 + \delta[w])$
```

Algorithm 5.5: *centrality*(G): Modified Brandes Betweenness Centrality

5.5.2 Newman Community Algorithm

Both divisive and agglomerative methods suffer from a common problem: when to stop adding or removing edges. Newman proposes the "modularity" function Q. Let ϕ_{ij} be the fraction of edges in the network that connect vertices in group i to group j. Groups, which we expand upon when presenting the algorithm, are the connected components of the graph. Then:

$$Q = \sum_i (\phi_{ii} - a_i^2) \tag{5.4}$$

where $a_i = \sum_j \phi_{ij}$. Intuitively, Q is the fraction of edges that lie within communities minus the expected value of the fraction of edges that lie within communities in a randomly constructed graph. As pointed out by Newman, a value of $Q = 0$ is indicative of community structure that is no more prevalent than would be expected if the graph were constructed randomly with the same degree as the original graph.

Newman's algorithm [28] for finding community structure is the following four-steps:

1. Determine edge betweenness centrality for all edges. Sort edges by non-increasing centrality.
2. While edges remain, remove the edge with the highest centrality. Ties are broken randomly.
3. Compute the modularity, Q, for each resultant graph.
4. Once all edges are removed, output the graph with the highest modularity score. This graph represents the inherent community structure in the original input graph.

Algorithm 5.6 presents the details of our implementation of Newman's algorithm. The algorithm takes as input the original graph edge adjacency matrix. It maintains two additional adjacency matrices, *pruned* and *best*. *pruned* corresponds with the original matrix edges minus the edges that have been removed thus far. *best* is used to retain the graph edges that produce the highest modularity score.

The algorithm first computes the edge betweenness centrality matrix, C_B, using our previous Algorithm 5.5. For each $e \in C_B$, add the 3-tuple $(i, j, betweenness)$, representing the node's numbers and betweenness score, into the set *edges*. The while loop repeats until no more edges remain. At each step, the edge with the highest betweenness centrality is removed and the *pruned* matrix is updated. For the pruned graph, we compute the set of connected components c using a standard connected components algorithm from e.g. [12].

```
$matrix[i][j]$, edge adjacency matrix
$best[i][j] \leftarrow matrix$, best adjacency matrix
$pruned[i][j] \leftarrow matrix$, pruned adjacency matrix
$C_B \leftarrow Centrality(matrix)$
$edges \leftarrow e = (i, j, betweenness)\ \forall e \in C_B$
while $edges$ not empty do
  $e \leftarrow pop\ max\{edges\}$
  $pruned = pruned - e$
  $c \leftarrow connectedComponents(pruned)$
  $compEdges[|c|][|c|] = componentEdges(c)$
  $q \leftarrow 0$
  for $i = 0$ to $|c| - 1$ do
    $a \leftarrow 0$
    for $j = 0$ to $|c| - 1$ do
      $a \leftarrow a + components[i][j]/numEdges$
    $\phi_{ii} \leftarrow components[i][i]/numEdges$
    $q \leftarrow q + (\phi_{ii} - a^2)$
  if $q > bestQ$ then
    $best \leftarrow pruned$
    $bestQ \leftarrow q$
$c \leftarrow connectedComponents(best)$
return $(c)$
```

Algorithm 5.6: $community(G)$: Newman Community Structure

For every component in c, we form a new matrix, $compEdges$ of size $|c|$ by $|c|$ that represents the number of edges in the *original* graph between components. Thus, $compEdges[x][y] = z$ means that there are z edges in the original graph connecting components x and y. Note that we number components randomly such that x has no particular significance other than identifying a particular component in the set of connected components. Algorithm 5.7 shows the procedure for creating the $compEdges$ matrix.

```
for $i = 0$ to $nodes - 1$ do
  for $j = 0$ to $nodes - 1$ do
    if $matrix[i][j] = true$ then
      $o[c[i]][c[j]] \leftarrow o[c[i]][c[j]] + 1$
```

Algorithm 5.7: $componentEdges(c)$: Find Community Edges

Finally, we iterate through the $compEdges$ matrix to compute the modularity score Q. If the current Q score is greater than any seen previously, we retain the score and the adjacency matrix that generated that score. The algorithm ends by returning the set of connected components for the adjacency matrix of the best modularity score. We refer the interested reader to [28] for several examples of practical applications of the Newman community algorithm.

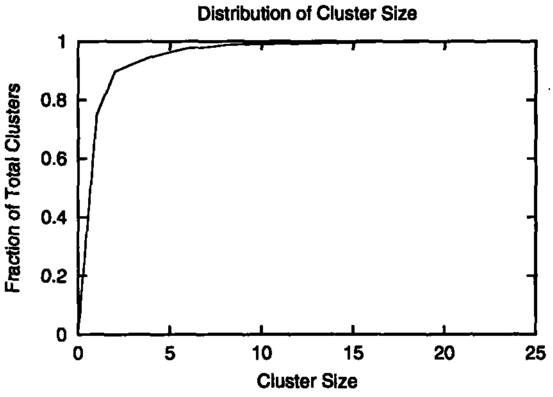

Figure 5-16: CDF of Cluster Size in ϵ-Equilibrium Network

5.5.3 Community Structure Results

We use the community algorithm to find the inherent community structure in our reorganized topologies. There are two motivations for this analysis. First, we are interested in whether the data we captured from the live Gnutella network contains a large enough number of nodes to fully exploit the ability of our system to reorganize around interests. With too few nodes, there may not be enough commonality to realize the full benefits of reorganization due to the bartering issues previously discussed. Second, the community structure of our resulting topologies reveals properties of the reorganization and provides novel insight beyond the simple degree distribution.

Figure 5-16 shows the distribution of cluster (community) sizes in the $\epsilon = 0.1$ equilibrium network. We see that approximately 50% of the clusters contain a single node, implying that these nodes are not central to the topology. This is not to say that they do not contribute to the aggregate positive utility, although naturally the isolated notes in the original topology will be in a single cluster as well, but rather that there are a small number of nodes which provide a central transit point.

In future research we wish to analyze in detail the properties of the clusters, for instance inter-cluster similarity metrics. Of particular interest is the nature of community structure in topologies where nodes propagate queries with higher TTLs.

5.6 Summary

In this Chapter we provided an extensive analysis of our interest-based reorganization algorithm, including comparisons against the existing Gnutella architecture and hypothetically optimal network topologies. We simulated the base and individually rational models of SuperPeer-based architectures and use the obtained global utility as a baseline.

Simulation of our algorithm shows that it reaches the predicted optimal utility while providing fairness not realized in other systems. However, the global utility falls far short of that obtained by, for instance, the individually rational SuperPeer network. As a potential remedy, we examined the possibility of including Bloom filters on all nodes to eliminate the bartering disbalance that causes so many nodes to fully disconnect. We find that such a network is able to produce comparable global utility as with the SuperPeer baseline.

A second analysis included epsilon equilibrium where we attempt to more accurately model the actual reward function of nodes. We found that by employing such a model, the disconnection problems of our other simulations disappeared and our reorganization algorithm found a solution where over 60% of the nodes are connected. Most encouraging is that this model converged to a utility 34% greater than achieved in the SuperPeer models while making no assumptions on the benevolence of nodes or centralized organization.

Finally, we presented an analysis of the community structure inherent in our resulting reorganized topologies. We evaluated the betweenness centrality and modularity of various networks. This analysis provides a novel angle of insight into understanding how much community structure exists in actual P2P networks.

...thinking how man alone of all creatures deliberately atrophies his natural senses and that only at the expense of others; how the four-legged animal gains all its information through smelling and seeing and hearing and distrusts all else while the two-legged one believes only what it reads.

- William Faulkner

Chapter 6

Conclusion and Future Work

In this thesis, we presented an architecture inspired by several trends that pose impending challenges to the construction of logical overlay networks. To date, the majority of overlay research has focused on algorithmic efficiency and purely technical issues, ignoring the policy and incentive problems that plague real systems. Of particular focus in this work is the optimality and fairness of systems in the presence of selfish nodes.

We rigorously defined ideal and fair networks and developed a continuum of optimality measures. We examined current Peer-to-Peer (P2P) overlays, both structured and unstructured, and found several weaknesses. First, their reliance on altruistic users in an increasingly competitive and hostile Internet, where free-riders and malicious nodes abound, makes the architecture vulnerable. Second, they treat all users equivalently, regardless of their interests, ignoring rather than exploiting the inherent user heterogeneity. In addition, while users have the ability to change their behavior, for example by moving around the system or tailoring their sharing preferences, the existing architectures place these functions outside of the system.

Our work presented an alternative architecture, based on the region abstraction, that brings the user's ability to affect the network and their incentives within the system. As such, our architecture is no longer dependent on the assumed altruism of any other nodes in the system.

A significant contribution of this work was the trace collection and its use in validating assumptions, making design decisions and simulating our algorithm against real-world work loads. We devoted a chapter to these traces, gathered from a portion of the Gnutella network, including extensive analysis of node behavior and file replication. To drive our algorithm design, we examined patterns of similarity in P2P nodes using accepted information retrieval techniques. As such, our research validated several commonly held assumptions.

We examined three possible architectures: the current Gnutella-like unstructured architecture, an augmented design with leaf nodes that reorganize around the SuperPeers and a homogeneous network of nodes that self-organize in a distributed fashion without relying on the altruism of any players in the network, in particular the SuperPeers.

We looked at the importance of Bloom filters in the existing architecture and analyzed the false positive rate as derived from our collected data. From simulation runs, we found, not unexpectedly, that the current architecture is highly dependent on the filtering capability and the willingness of the SuperPeer network to absorb the majority of the query burden.

We then tackled the crux of our thesis, a world in which SuperPeers no longer exist or are untenable for whatever reason. We described several problems and rationale for our design decisions. For example, in response to uncooperative nodes in the system (bullies), we used a hysteresis mechanism based on simulated annealing. We also found that actions cannot be made in isolation, but rather amongst coalitions of the existing neighbor set. Our design included a discussion on the selection of the utility function and its shape and discount factor α.

Next, we looked at the problem of finding optimal graphs and proved the fact that enumerating the entire set of possible graphs is exponentially hard. We used this result to develop an optimal topology search algorithm. The optimal topologies are used to as a basis against which to compare the performance of reorganized topologies.

We examined reasons why our distributed selfish algorithms may not perform as well as the optimal topologies. We introduced four reasons for utility suboptimal self-reorganizing networks: anarchy (selfish behavior), indifference, myopia and ordering to give further basis for our subsequent analysis and results.

We then evaluated the performance of our architecture against our collected real-world work load. Our objective was to show the current level of utility and happiness achieved with the existing architecture. We then systematically tore down the implicit assumptions of altruism built into the existing architecture while showing the resulting negative impact on utility. Once we had a selfish equilibrium, with much lower global utility, we showed the ability of our algorithm to reorganize and restore the utility of individual nodes and the system as a whole to similar levels as realized in the SuperPeer network.

Simulation of our algorithm shows that it reaches the predicted optimal utility while providing fairness not realized in other systems. However, the global utility falls far short of that obtained by, for instance, the individually rational SuperPeer network. As a potential remedy, we examined the possibility of including Bloom filters on all nodes to eliminate the bartering disbalance that causes so many nodes to fully disconnect. We found that such a network is able to produce comparable global utility as with the SuperPeer baseline.

A second analysis included epsilon equilibrium where we attempt to more accurately model the actual reward function of nodes. We found that by employing such a model, the disconnection problems of our other simulations disappeared and our reorganization algorithm found a solution where over 60% of the nodes are connected. Most encouraging was that this model converged to a utility 34% greater than achieved in the SuperPeer models while making no assumptions on the benevolence of nodes or centralized organization.

Finally, we presented an analysis of the community structure inherent in our resulting reorganized topologies. We evaluated the betweenness centrality and modularity of various networks. This analysis provided a novel angle of insight into understanding how much community structure exists in actual P2P networks.

There are many possible avenues of further research beyond this thesis. Our architecture hints at several possible improvements including:

- *Intelligent connection:* In order to maximize the probability of finding a good neighbor, a node could request a list of neighbors from its current high-utility peers. These peers are also likely to have a high utility and are ideal candidates to connect with.

- *Strategic nodes:* Modify our algorithm to discourage not just selfish but also *strategic* behavior by nodes. A malicious node that provides false answers could be detected once the file cannot be downloaded. Answers from malicious nodes will decrease their utility and be additionally penalized by the system. Busy or congested nodes that wish to discourage connections and query traffic could drop queries as an additional action.

In order to lend tractability to our simulation and modeling of the network, we ignored several elements. An enhancement of our work might include:

- *System temporality:* Model dynamic queries over time. This would require gathering data from a much larger portion of the Gnutella network.

- *Utility function:* The discrete utility function is a simple construction and, as shown at numerous points, not necessarily representative. While it is not possible to define the "correct" utility function, we believe a more complex temporal function would better model reality. For instance, a continuous utility function that takes into account the instantaneous bandwidth of queries and downloads.

Finally, we are very interested in the applicability of our architecture to other problem domains. We believe our architecture, and the region abstraction, is a general construction that could be used in mobile and sensor networks, routing and web searching.

For example, consider searching for content on the world-wide web. We believe there are two factors driving web search. First, a need to search on larger and larger data sets. For instance the files a user's local machine, not just content published on the web. Second, a need to perform directed searches. The formation of a search network based on interests was first proposed in [15]. Our architecture could be used in a similar fashion such that the formation of an overlay devoted to content searching is driven by repeated queries.

Note that one of the big disadvantages with using locality to organize the network is due to the overhead and delay in establishing the network, i.e. learning the interests and finding appropriate attachment points. A second disadvantage is that the participants change quickly, possibly faster than the system can adapt. Web search is an interesting additional problem domain because it represents a constant stream of queries (every person continually searches!) and is therefore a natural source for directing the formation of the network. Local agents on each computer could be used instead of the current client to search-engine server model.

We hope that the work and results reported in this thesis help us and others to answer these outstanding questions.

Bibliography

[1] E. Adar and B. Huberman. Free riding on Gnutella, 2000.

[2] A. Asvanund, S. Bagla, M. Kapadia, R. Krishnan, M. D. Smith, and R. Telang. Intelligent club management in peer-to-peer networks. In *Proceedings of First Workshop on Economics of P2P*, 2003.

[3] H. Balakrishnan, F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in P2P systems. *Communications of the ACM*, February 2003.

[4] N. Belkin and W. Croft. Retrieval techniques. 22:109–145, 1987.

[5] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, July 1970.

[6] U. Brandes. A faster algorithm for betweenness centrality. *Mathematical Sociology*, 2001.

[7] Y. Chawathe, N. Lanham, S. Ratnasamy, S. Shenker, and L. Breslau. Making gnutella-like P2P systems scalable. In *Proceedings of ACM SIGCOMM*, 2003.

[8] N. Christin, J. Grossklags, and J. Chuang. Near rationality and competitive equilibria in networked systems. Technical Report TR-2004-04-CGC, University of California at Berkeley, 2004.

[9] B Chun. Structured peer-to-peer network optimization game. In *Proceedings of First IRIS Student Workshop*, August 2003.

[10] B. Chun, R. Fonseca, I. Stoica, and J. Kubiatowicz. Characterizing selfishly constructed overlay routing networks. In *Proceedings of IEEE INFOCOM'04*, 2004.

[11] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46, 2001.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[13] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for P2P systems, 2002.

[14] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica. Free-riding and whitewashing in peer-to-peer systems. In *Proceedings of ACM SIGCOMM PINS Workshop*, 2004.

[15] Paul Francis, Takashi Kambayashi, Shin ya Sato, and Susumu Shimizu. Ingrid: A self-configuring information navigation infrastructure. In *4th International World Wide Web Conference*, 1995.

[16] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40:35–41, 1977.

[17] Gnutella. Gnutella: Distributed information sharing, 2000. `http://gnutella.wego.com`.

[18] P. Krishna Gummadi, S. Saroiu, and S. Gribble. A measurement study of napster and gnutella as examples of peer-to-peer file sharing systems. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, 2002.

[19] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[20] Brad Karp, Sylvia Ratnasamy, Sean Rhea, and Scott Shenker. Spurring adoption of DHTs with OpenHash, a public DHT service, 2004.

[21] P. Keleher, B. Bhattacharjee, and B. Silaghi. Are virtualized overlay networks too much of a good thing, 2002.

[22] Steven P. Ketchpel. Forming coalitions in the face of uncertain rewards. In *National Conference on Artificial Intelligence*, pages 414–419, 1994.

[23] R. Krishnan, M. Smith, Z. Tang, and R. Telang. Impact of free-riding on peer to peer networks. In *37th Hawaiian International Conference on System Sciences*, 2004.

[24] R. Krovetz. Viewing Morphology as an Inference Process,. In *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 191–203, 1993.

[25] Ji Li. Improving application-level network services with regions. Technical Report MIT-LCS-TR-897, Massachusetts Institute of Technology, 2003.

[26] Sharman Networks Ltd. KaZaA media desktop, 2001. `http://www.kazaa.com`.

[27] Mutella. Mutella, 2003. `http://mutella.sourceforge.net`.

[28] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev.*, 2004.

[29] T.-W. J. Ngan, D. S. Wallach, and P. Druschel. Enforcing fair sharing of peer-to-peer resources, February 2003.

[30] Christos H. Papadimitriou. Algorithms, games, and the Internet. *Lecture Notes in Computer Science*, 2076:1–??, 2001.

[31] W.H. Press. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[32] R. Radner. Collusive behaviour in noncooperative epsilon-equilibria of oligopolies with long but finite lives. *Journal of Economic Theory*, 22:136–154, 1980.

[33] E. Rasmussen. *Clustering algorithms*. Prentice Hall, 1992.

[34] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. Technical Report TR-00-010, University of California at Berkeley, 2000.

[35] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4)*. Internet Engineering Task Force, 1995. RFC 1771.

[36] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 6(1), 2002.

[37] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–??, 2001.

[38] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks, 2002.

[39] SETI. Search for extra-terrestrial intelligence at home. `http://setiathome.ssl.berkeley.edu/`.

[40] L.S. Shapley. *A value for N-person games.* Princeton University Press, 1953.

[41] Sumeet Singh, Sriram Ramabhadran, Florin Baboescu, and Alex Snoeren. The case for service provider deployment of super-peers in peer-to-peer networks, June 2003.

[42] A. Singla and C. Rohrs. Ultrapeers: Another step towards gnutella scalability, 2002. `http://www.limewire.com/developer/Ultrapeers.html`.

[43] K. Sollins. Regions: A new architectural capability for networking, August 2001. White paper.

[44] K. Sollins. Designing for scale and differentiation. In *Proceedings of ACM SIGCOMM FDNA Workshop*, 2003.

[45] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems, 2003.

[46] A. Steger and N. Wormald. Generating random regular graphs quickly, 1999.

[47] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[48] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, University of California at Berkeley, 2001.