ABSTRACT

ZHANG, QINGHUA Improving performance of peer-to-peer systems by caching. (Under the direction of Dr. Douglas Reeves).

Recently, Peer-to-Peer (P2P) has attracted a great deal of interest in industry and research literature. P2P systems are application layer networks, in which logically distinct computing elements - peers, bear comparable roles and responsibilities. P2P enables peers to share resources in a distributed manner. Existing P2P systems work well but are inefficient with respect to information retrieval. Some measurements show that more than half of the current Internet traffic is P2P traffic. Some search methods currently used by P2P groups flood the network, thus consuming a lot of bandwidth. In addition, some P2P applications require some forms of global knowledge of peer resources.

Caching is one way to improve the performance of any system that makes repetitive requests. This thesis proposes a selective query-forwarding scheme based on caching. This simple caching mechanism improves efficiency and scalability in information retrieval for P2P systems. Query processing is expedited by caching similar queries or replies, thus making searches more efficient.

The performance of this caching-based search algorithm is evaluated and compared with two existing P2P search algorithms (flooding and Random Walk) in P2P file sharing systems. The simulation experiments are designed and performed based on some measurement and empirical data. The results show this caching-based scheme is an attractive technique for keyword based searching in P2P systems. In some cases it achieves 75% query hits through caching. Its performance is also superior in that it consumes less bandwidth and takes less time to satisfy queries. Finally, this approach doesn't incur additional network traffic to develop knowledge on resource location and thus scales well with the size of the network.

Improving performance of peer-to-peer systems by caching

by

Qinghua Zhang

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial satisfaction of the
requirements for the Degree of
Master of Science

Department of Computer Science

Raleigh

2003

Approved By:

Dr. David Thuente

Dr. Khaled Harfoush

Dr. Douglas Reeves
Chair of Advisory Committee

To my parents,

Yi Zhang,

Xiaoling Wu

and my sister,

Wenyan Zhang

Biography

Qinghua Zhang was born in Jiang Xi, P.R.China, in 1977. She received her Bachelor's degree in Computer Science and Engineering from Beijing University of Posts and Telecommunications, Beijing, P.R.China in 1999. From 1999 to 2001, she continued her graduate study in Beijing University of Posts and Telecommunications. Since 2002, she has been a Masters student in the Department of Computer Science at North Carolina State University.

Acknowledgements

My thesis advisor, thesis committee members, many friends and colleagues, and my family have contributed to this dissertation and played a significant role throughout this work. Without any of them, this work could be very difficult to accomplish. I thank all of you for your help, motivation and constructive criticism. North Carolina State University's Cyber Defense Laboratory provided an ideal working environment.

First, I would like to thank Dr. Douglas Reeves for his support during this work. His guidance and all the fruitful comments and suggestions laid the foundation of the research presented in this thesis. It has been a pleasure working with Dr. Douglas Reeves in the development of this thesis and I cannot thank him enough. I also thank Dr. Khaled Harfoush for his good advice, insightful comments and valuable feedback. Finally I would like to thank Dr. David Thuente for taking time out of his busy schedule to be on my thesis committee.

Many friends and colleagues have helped me in various ways throughout the entire duration of this thesis. They are - Ranjana Rao, Jing Teng, Amit Sawant, Pan Wang, Dingbang Xu, Donggang Liu and others whom I may have mistakenly forgotten to mention. I am grateful to Ranjana for her helpful discussion with me about my research ideas in this thesis. And I also thank her for providing me the thesis latex template. Jing, thanks for helping me in debugging the errors in my code. I thank Amit for his time in proofreading my thesis draft. Thanks go to Pan, Dingbang, Donggang for their helping me with debugging my program and using Latex for writing my thesis.

I am truly indebted to my family for their love, patience, encouragement, and blessings to me. I sincerely thank my Dad , Yi Zhang and my Mom, Xiaoling Wu for their all along continuous support, trust and beliefs in me in pursuing my objectives. I own them my beliefs in education and my willingness to learn and success on going through the requirements of the Master Degree. Last I would like to thank my sister, Wenyan Zhang, for her good wishes and moral support. I affectionately dedicate this thesis to them. Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recently, the popularity of Peer-to-Peer applications has grown dramatically, especially in the field of multimedia file sharing such as Napster[1] and Gnutella[2]. This has resulted in a flurry of research in peer-to-peer systems such as Chord[3], CAN[4], Pastry[5], Kazaa[6] and many others.

1.1 What is Peer-to-Peer?

Peer-to-Peer (P2P) computing is different from traditional Client-Server infrastructure and is defined as sharing of resources and information through direct exchange. In traditional client-server applications, the client requests resources or services and the server provides the desired resources or services. However, in P2P computing, peers have comparable roles and responsibilities and can act as either client or server. For example, in distributed multimedia file sharing, any peer can request a multimedia file (like .mp3) and any peer may provide the requested file. P2P networks allow two or more machines to pool their resources, whereas in client-server networks, resources are stored on a centralized file server which may have tens of thousands of clients. Centralized file servers are prone to malicious attack and suffer performance bottlenecks at times when many clients flood the server with requests for resources at the same time. Moreover, information systems are further degraded when they contain broken links, which refer to out of date information.

However, in P2P networks, peers control resources as they speak to each other directly and without intermediate or central authority. Peers can provide back up service for each other. In contrast to Client/Server infrastructures, in peer-to-peer networks, isolated node failure can be quickly identified and bypassed. The other benefits of P2P systems are: adaptation, fault-tolerance, and high availability through massive replication of single resources.

1.2 Origin of Peer-to-Peer

Though P2P applications gained popularity in recent years, P2P computing is not a totally new paradigm. According to [7], USENET (born in 1979) is a very successful, completely decentralized network of peers. It is a distributed application, which provides its newsgroup to worldwide readers. The most significant trends in P2P application research are decentralizing of software engineering by employing such technologies as Business-to-Business (B2B) transactions [7] and the availability of more powerful computing capabilities on the Internet because of decreasing cost of bandwidth, thus making large scale, distributed computing a reality.

1.3 P2P applications and research challenges

There are a lot of peer-to-peer applications. Peers can share the excessive CPU processing cycle Seti@Home[8]; possess storage to perform large scale distributed computing [9]. According to [10], P2P can also be applied to: collaboration Groove[11] and infrastructure systems [12]. It could be used for distributed communication applications such as AOL instant Messenger and ICQ. Additional recommended applications for P2P can be found at O'Reilly P2P directory service [13].

File sharing is one of the most popular applications. However, there are a lot of research challenges still to be addressed in P2P file sharing systems, including resource management, security, content location, performance issues, standardization and interoperability [14].

This thesis addresses one of the fundamental challenges facing P2P systems-how to locate desired files efficiently. Many solutions have been proposed for decentralized P2P file sharing. Since decentralized computing is the trend, this paper does not concern itself with the details of centralized computing solutions, like Napster [1]. P2P systems can be categorized into two major classes: structured and unstructured. Unstructured P2P file sharing systems, like Gnutella [2], have no centralized directory and no precise control over network topology and file placement. The systems are formed in an ad-hoc way. In the unstructured P2P system, peers broadcast queries-requests for desired files-to neighbor peers. Peers may respond to this query by supplying the desired file to the requestor or by forwarding the query to their neighbors if they cannot supply the desired file themselves. The query floods in this overlay network until it travels a certain radius or hops. Structured P2P file sharing systems like Chord [3], CAN [4], Pastry [5] have no central directory but they use Dynamic Hash Table (DHT) technology to tightly control the formation of the network and how files on the network are managed. Structured P2P systems employ a complex structure to make search and information retrieval efficient, which also means that this search method generates fewer messages than the flooding method and consumes less bandwidth.

1.4 Other Issues about P2P application

There are some issues other than performance in P2P research. Many people object to P2P research because they believe that P2P technologies provide too much benefit and convenience to "pirate" businesses. Napster, which was shut down in 2002 because it violated copyright law, is the most frequently cited case in such arguments. To overcome this kind of resistance, measures will have to be taken to prevent illegal usage of P2P technologies. Napster was revived after changing its practices so that it now charges users for downloading music. Such ethical considerations are outside the scope of this thesis.

1.5 Research Goals and Contributions of this Thesis

Simplicity, self-organization, the absence of a complex control structure, fault-tolerance and high availability make Gnutella-like networks widely applicable [13]. According to the evaluation criteria in [39], a good search algorithm must be scalable, efficient, and responsive. That is, a search algorithm is efficient if it does not overwhelm network bandwidth by generating a huge number of redundant messages in an uncontrolled fashion

[39]. The flooding based search algorithm used by Gnutella-like systems is inefficient, so that these systems suffer from poor scalability and are prone to cause great traffic, which can be as high as 60% of Internet traffic [15]. Improvements to the search mechanisms continue, including random walk [16], expended ring [16] and cumulative learning [17,18]. This thesis work also aims at improving the pure flooding search algorithm in Gnutella-like systems by exploring the locality principle of queries. Caching is an effective way of improving the performance of any system that makes repetitive requests. Although some recent papers [19, 20, 21, 22, 23, 24] discuss this aspect, little work has been done to evaluate the performance of caching in P2P systems. This thesis proposes a selective query-forwarding scheme based on caching which is an efficient, scalable yet simple mechanism for improving information retrieval problem in P2P systems. This approach is efficient since query processing is expedited by caching similar queries or replies.

The performance of this caching-based search algorithm is evaluated and compared with two existing P2P search algorithms-flooding and Random Walk-in P2P file sharing systems. The simulation experiments are designed and performed based on some measurement and empirical data. The results show this caching-based scheme is an attractive technique for keyword searching in Peer-to-Peer systems. In some cases it achieves 75% query hits through caching. Its performance is also superior in that it consumes less bandwidth and takes less time to satisfy queries. Finally this approach doesn't incur additional network traffic to develop knowledge on resource location and thus scales well with the size of the network.

1.6 Thesis Organization

The thesis is organized as follows. Chapter 2 discusses the background and related work on P2P, mainly focusing on different techniques in information retrieval and measurement studies. Chapter 3 presents the thesis scheme of improving search based on caching in P2P systems in detail. Chapter 4 describes the simulation design for validating the proposed scheme. Simulation results are presented and evaluated in Chapter 5. Chapter 6 concludes this thesis.

Chapter 2

Background and Related Work

In this chapter, major architecture of P2P systems and a number of different search techniques in P2P networks are introduced. In section 2.1, several types of P2P architectures are discussed and compared. Section 2.2 studies a number of different search techniques in P2P networks. Major advantages and disadvantages are analyzed. Section 2.3 introduces the major effort in measuring practical P2P systems' properties and performance behaviors.

2.1 Background

Peer-to-Peer networks have been studied a lot in the past few years. There are several different types of architectures for P2P networks such as: Centralized vs. Decentralized; Structured vs. Unstructured. Also, there are some hybrid architectures. A recent paper by P. Ganesan[25] proposes a Decentralized hybrid (partly structured and partly unstructured) P2P network *Yappers*. Another hybrid architecture Kazaa[6] uses superpeers, which serve the neighboring peers.

Centralized: Napster [1] is a typical centralized P2P network. It has a central directory server hosting an index to locate peers' files. When registering into the P2P system, a peer needs to tell the server what files it would like to provide to share with other peers. This directory is constantly updated. When peers (nodes) in the P2P networks generate requests for specific files, they deliver the requests to the central directory server, looking for the peers that hold the desired files. But, centralized approaches suffer from poor scalability and the failure of a single point will cause the whole system to fail. So a lot of effort has been put into decentralized P2P research, where query processing and file transfer are fully distributed.

Decentralized Structured: It is commonly referred to as Distributed Hash Tables (DHTs) [3,4,5]. In this system, the overlay topology is tightly controlled and the peers joining the system have unique identifiers. File locations in the peers are determined by some hashing algorithms. This highly structured P2P design has been of interest (or popular) in the research literature. However, no such architecture is currently applied in the real networks. In order to control and maintain the network topology based on DHTs, the systems need overhead to re-organize the network. The re-organizing overhead is especially high when the system exhibits a great dynamic property, e.g., peers constantly joining and leaving; bulk of new files being generated and old files expiring and deleting. Moreover, due to DHTs, the systems cannot efficiently support partial-match file lookup [27], which is a key reason for it not being widely adopted in real application.

Decentralized Unstructured: Gnutella[2] is a typical example of decentralized unstructured system. In contrast with structured P2P systems, unstructured P2P systems neither have strict control over network topology nor file placement. Peers join the network and form the topology in accordance with some loose rules [28]. There is no central directory server to index data items as centralized P2P systems do. Rather, data items or files are locally indexed and stored. The query can only be resolved by the peers, which hold the file. Flooding is a typical way for queries to locate the peers holding the desired files. Unstructured P2P systems, in contrast to structured systems, are very resilient to peers joining and leaving the system. However, due to the flooding nature, the system suffers from poor scalability and always results in large traffic on the network and adds much burden on the network participants.

Decentralized Hybrid structured: This architecture design objective is to combine the strength of both the unstructured and structured. According to a recent paper [25], for each peer in the search network, the system builds a small DHT consisting of nearby peers and then provides an intelligent search mechanism that can traverse all the small DHTs. It can be built on an arbitrary overlay while retaining DHT-like search efficiency. This design doesn't try to impose topology constraints and optimizes partial lookups and only contacts nodes that can contribute toward the search result and minimize the maintenance overhead.

Hybrid architecture: This architecture uses supernode. Kazaa[6] is a typical application based on Fasttrack Technology [29]. Kazaa specially uses designated supernodes that have higher bandwidth connectivity. Each supernode stores pointers to each neighboring peer's files. All queries are routed to the supernodes, which in turn can look for the desired files index in its storage.

This thesis aims at improving search in Gnutella-like decentralized, unstructured P2P systems, which is used by a large community of Internet users [13]. Moreover, there is still room for improving the poor performance caused by the flooding mechanism.

2.2 Related Works

A lot of work has been done to improve the Gnutella-like systems' performance and scalability. According to [30], the search method can be categorized as either blind or informed. In blind search methods, peers neither hold any information related to document locations nor use heuristic hints to search for specific peers which are highly likely to hold the desired documents. It just tries to propagate the query to a sufficient number of nodes in order to satisfy the request [30]. Instead, in an informed search, a centralized or distributed service helps in the search of the requested files by offering document location information, or heuristic hints, e.g. learning from the past history.

2.2.1 Blind Search Methods

- 1. **Pure Flooding** [2]: The original Gnutella algorithm uses flooding to deliver the queries for files or objects. When the query is delivered, the peer will contact all its neighboring peers to forward the query. Thus the query will flood in the network until it travels a certain radius (It will terminate when the TTL value in the query packet is reduced to 0). This method is simple and can discover maximum number of copies of the file or object. However, it generates huge overhead and traffic in the network. Thus it has a very poor scalability.

- 2. **Random Walk** [16]: Instead of forwarding the query to all the neighboring peers in original Gnutella [2], in random walk method, a peer will only forward the query message to k randomly chosen neighbors. A query message is a walker that follows its own path. A walker will terminate if there is a data hit or if a failure is determined by TTL-based method or checking method. In the TTL-based method, if TTL value in the query packet is reduced to 0, the query will be discarded and won't be forwarded again. In the checking method the walkers periodically contact with the query source asking whether it possesses the desired files or not so that it can stop traveling. The most important advantage of this method is that it greatly reduces the number of messages traversing in the network. Simulation results in [16] show more than one order of magnitude number of messages are reduced, compared to pure flooding scheme. However, the most serious disadvantage of this method is that the performance is highly variable. In the random walk method, such cases are common: there exist files that one requests, but due to unforeseeable random choice and different topology of the network, the search will always fail. However, flooding scheme hardly has such problems. According to [42], there are two problems associated with random walk. One is that random walk method doesn't take into account any indication of how likely it is that a random chosen node will have responses for a query at each step. The other problem is that a random walker query may arrive at a node that is already overloaded with traffic and thus it may get queued for a long time before it is handled. An improvement on the first problem could learn on previous experience in order to make a better choice of the neighbors to which the queries are forwarded. This thesis work provides a solution to this problem. [42] proposes a dynamic topology adaptation protocol to solve the second problem. The basic idea is to put a node's capacity into consideration when a query is to be forwarded.

- 3. **Modified-BFS** [18]: This method tries to improve the pure flooding method by reducing the number of query messages in the network. This variation of pure flooding scheme allows peers to choose only a ratio of their neighbors to which the queries are forwarded. This method does reduce the number of messages generated, compared to pure flooding, but still a lot of messages are circulated in the network.

- 4. **Expending Ring** [16]: This method aims at addressing TTL selection problem by using successive floods with increasing TTL values. A peer starts a flood with a small TTL value and waits to see whether the search is successful or not. If successful, then the peer stops flooding, otherwise, the peer increases the TTL value and another flooding trial begins. This method can significantly reduce the message overhead . The expending ring achieves better results when the object's replication ratio increases. Those savings are at the expense of increase in the delays to find the object. The disadvantage is that when the search termination condition relates to a user-defined number of hits becomes more strict, this method will produce even bigger loads than the standard flooding scheme since it will try flooding many times, instead of reducing the number of messages.

- 5. **Super-Nodes** [6, 31, 29]: This method designates supernodes that have higher bandwidth connectivity. Each supernode stores pointers to each neighboring peer's files. All queries are routed to the supernodes, which can look up for the desired files' index in its storage. This approach seems to have better scaling than Gnutella [2]. However, no detailed documentation and measurement to this system Kazaa[6] are available to the public. [31] also uses caching scheme in supernode to help reduce the traffic. It relies on supernode to cache files' routing information and information of all files in this supernode domain. If the primary supernode finds the requested file is cached in its domain, it will reply the originator with the id of the peer which stores the copy and that peer can satify the originator with the requested file. These two cache methods (routing cache and file cache) have an obvious shortcoming which counteracts the effectiveness of using supernodes and P2P systems' high avaiability: the requests for the same file will always be redirected to the same peer. However, this is not a problem in the caching scheme proposed in this thesis.

2.2.2 Informed Search Methods

- 1. **Interest-Based Shortcut** [26]: This approach explores the principle of interest-based locality. If a peer has a particular piece of content that one is interested in, then it is likely that it will have other pieces of information that are also of interest [26]. Based on this principle, peers form a loose interest-based structure on top of the Gnutella network. Every peer will begin the search by first looking at the peer whose shortcut it is pointing to. If it fails, it will still follow the flooding method in Gnutella network. This method can improve the performance of flooding-based Gnutella system. However, when users exhibit different patterns of file or object collection and object requests (e.g, the interest is constantly changing), this method could worsen Gnutella's performance.

- 2. **Adaptive Probabilistic Search** [17]: Each node deploys k different walkers for object discovery and performs probabilistic instead of random forwarding. Each peer delivers the query to one of the neighbors with probability given by the local index kept for each object that it has requested per neighbor. The value of the index indicates the relative probability of this neighbor node to be chosen as the next hop in the future requests for specific objects and is updated using feedback from the previous searches. If a search succeeds (fails), the relative probabilities of the nodes about the walker's path are increased (decreased). APS is bandwidth-efficient, robust and adaptive in rapidly changing environments, because of the algorithm's learning nature.

- 3. **Keywords-based Query Route table** [32]: This scheme tries to avoid broadcast queries on the Gnetella network. The key idea is to have each host maintain a query route table by hashing file keywords and regularly exchanging them with their neighbors. To minimize the overhead of exchanging tables, standard compression techniques (bitmap, etc) are applied. In order to implement this scheme for Gnutella Network, they extend Gnutella protocol by proposing an extensible message - ROUTE_TABLE_UPDATE. Their scheme has potential to reduce the Gnutella bandwidth requirement, but they still need to do simulation to evaluate this scheme before it is widely deployed.

- 4. **Push-Pull Gossiping** [33]: It expands the P2P system scope to include "communities". These communities can be interest groups. They can overlap. Pull-push gossiping aims at improving information search within a P2P community. Such P2P network exhibits property of a small-world, scale free network. A discovery phase is initialized to gather data on peers that would be interested in receiving certain information. Through this data, highly influential peers (called seers) are identified and in push phase gossiping, information is multicasted to these seers. In pull phase gossiping, peers can easily and quickly retrieve this information from a nearby seer.

- 5. **Routing Index (RI)**[34]: This scheme allows nodes to create and maintain routing index for each of its neighbors. If a peer cannot answer a query, it will forward the query to neighbors that are most likely to have answers (with highest "goodness" value). Three RI schemes are proposed to rank the out-going links according to the expected number of documents from every category that can be found through each out-link. This is a "keyword-based" search method. However, this RI approach isn't suitable for highly dynamic environment since the information on the documents needs to be exchanged through flooding in order to create and maintain RIs

- 6. **Local Index** [35]: The basic idea is to reduce the number of nodes that process a query. Each node maintains an index over the files stored at all nodes within a certain radius or hops of itself and can answer the queries based on the behavior of every node within this radius. This method has high accuracy and hit ratio, and the processing time and response time is smaller as fewer nodes will get involved in processing the query. However, this method has a big overhead, which is comparable to the pure flooding scheme since it needs to create and maintain the indices at each node, especially in a highly dynamic environment. In order to maintain the information, the scheme needs to flood with TTL=r whenever a node within r hops joins/leaves the network.

- 7. **Intelligent-BFS** [18]: This is similar to APS [17] in the sense that it also uses feedback to guide the search. Each peer keeps a profile for each of its neighbors, recording the most recent replies from or through that peer and uses this profile to rank these neighbors. When a query arrives, if the peer cannot answer it, it will identify all similar queries to the current one, and choose a set of its neighbors, which have returned the results for these queries, to forward the query to.

2.2.3 Measurement Studies

Besides the numerous improvements on P2P systems' search methods, there are some measurement studies of P2P infrastructures [36, 37, 20, 38, 19]. They try to characterize the basic properties of P2P file sharing systems. And these facts can be used to evaluate the P2P file sharing systems.

[36] performed a detailed measurement study on two popular P2P systems: Napster and Gnutella, especially based on bandwidth, latency, availability, and file-sharing patterns of these peers. Their study shows significant amount of heterogeneity in both systems and peers, and they tend to deliberately misreport information if there is incentive to do so. Moreover, although peers are designed with symmetry of responsibilities in mind, client-like and server-like behaviors clearly take in a significant fraction of systems' population.

[37] performed a measurement study on a currently popular P2P system: Kazaa about the nature of its workload. The results show the objects that Kazaa users exchange, are large, immutable video and audio objects, causing the typical client to fetch any given object at most once. Also the popularity of Kazaa objects change over time. Their study shows "fetch-atmost-once" behavior of clients cause the aggregate popularity distribution of objects in Kazaa to deviate substantially from zipf curves. Another important conclusion is the existence of significant locality in the Kazaa workload, and therefore substantial opportunity for caching to reduce wide-area bandwith consumption. Although the proposed scheme uses cache to help improve the search in a different aspect, the basic principle is same: there is locality in the P2P workload.

[20] studied the nature of P2P traffic based on measurement on FastTrack-based P2P system, including Kazaa. It also concludes that caching of P2P traffic is an effective and desired means for dealing with the bandwidth drain caused by the increase of P2P traffic. [21] also demonstrates that sharing of cache among Web proxies reduces Web traffic.

[38] studied AT&T's backbone traffic by analyzing three popular P2P systems: FastTrack, Gnutella, and DirectConnect and it confirmed the results of [36], that the traffic distributions for traffic volume, connectivity, availability and average bandwidths usage are extremely skewed. All the three P2P systems' hosts exhibit high dynamics and only a small fraction of hosts are persistent over long periods.

[19] studied the locality issue of files in Napster and Gnutella. Their measurement study indicates there is significant locality in the stored and transferred files on Napster and Gnutella. They concluded that caching can improve the performance of these systems. [24] detects the locality in search engine queries and implies caching search results could improve search performance. Their work confirms the approach of caching on queries and

replies to improve the performance of P2P file sharing system.

2.3 Summarization

This chapter talks about major, current P2P systems architectures, some search techniques and measurement work which are the major source of reference to this thesis's simulation work. Few of these seach techniques discover the locality principles of queries. A new search approach which uses caching mechanism has been proposed and is described in the coming chapter. Chapter 3 presents the new approach in detail.

Chapter 3

Selective Query-forwarding Scheme based on Caching

This chapter presents a caching-based selective query-forwarding scheme, which is a simple yet very effective mechanism for information retrieval in P2P networks. The objective of this algorithm is to help peers to quickly and efficiently find the most relevant answers to queries.

The key to improving the speed, efficiency and accuracy in query processing is to make use of the locality principles of queries. By caching similar incoming query messages and reply messages, a peer grasps global knowledge on the location of information resources, and thus greatly expedites its access to needed information without a lot of unnecessary query message forwarding.

3.1 Motivation

P2P file sharing has popular applications [1, 2, 6] and has generated a lot of research interest in recent years [3, 4, 5], but there are still many challenges that must be addressed by further research. Efficient file location is one of the most important research issues. Numerous schemes have been proposed. DHT based P2P file sharing systems, though very efficient in locating objects, require strict controls on file placement and topological construction and the need for controlling and maintaining network structure in a highly dynamic environment causes excessive overhead.

Gnutella is a very popular P2P file sharing application, however, its flooding-based file search mechanisms result in poor scalability and cause a great deal of unnecessary traffic on the Internet. This thesis aims to design a new scheme that is simple, efficient and scalable. A lot of excellent work has been done in this field. This thesis work concentrates on two aspects that have not received adequate attention in research; using the locality principle of queries and file sharing and using caching techniques to reduce network traffic during the search process. The method in this thesis doesn't need to exchange additional messages with each other to maintain network structure or knowledge of files' location. From the aspect of network scalability, this method is suitable in a highly dynamic environment.

3.2 Basic Idea

The basic idea is to expedite the search process by changing the way that query messages are forwarded, so that queries are routed first to those peers who are most likely to answer the query without incurring additional message exchanges in the system. This reduces the traffic problem caused by P2P query-message flooding which occurs in Gnutella-like systems and makes the P2P system more scalable. In the scheme proposed by this thesis, these objectives are achieved through caching queries and replies and by selective-message forwarding of active queries. Queries received from other peers and replies received in response to queries, but not those from intermediate peers, are cached with the associated keywords and file types. When a new query is generated, first the cache is searched to identify similar queries that have already been processed. The query is then selectively forwarded to those peers who have made similar requests in the past. This preliminary search of the cache yields an expedited search, as the identified peers are more likely to provide the desired file or to give instructions on how to gain access to the requested information.

When a peer receives a new query, it will first look up its own file collection. If it finds a hit, it will simply reply to the requestor; otherwise, a search will be made of the peer's reply-cache-list and request-cache-list to see if that peer has processed a similar request in the past. If a query or reply is found in the cache that matches the incoming query, the message is forwarded to those peers identified with the cached query or reply; otherwise, this peer will randomly choose some neighbors who will receive the forwarded queries. For each peer, the cache contains requests generated by others and all the replies to its own requests. Forwarding queries to peers who have sent similar queries in the past is expected to yield more efficient searches because these peers are more likely to have a broad knowledge of the requested files.

The scheme proposed in this thesis addresses the two major concerns needed for its implementation; how to cache queries and replies and how to redesign the matching algorithm to make it more efficient. Two matching schemes are defined: exact-matching and similar-matching. Exact-matching means that the file includes all the keywords in the query and thus satisfies the query. Similar-matching means that two queries each may include all the keywords of the other.

According to [42], keyword searches are more prevalent and more important than exact name matching. There is no unambiguous naming convention for files in P2P systems [42], thus this thesis adopts keyword based caching. Similar keyword based query methods can also be found in recent peer-to-peer research work [32].

3.3 Requests- and Replies- Caching Heuristics

Cache mechanisms have been used in a variety of environments-OS high-speed cache, web proxy cache and P2P traffic caching-to improve systems performance. [21] notes that caching has been recognized as one of the most important techniques to reduce bandwidth consumption. This thesis proposes the caching of requests and replies to improve search efficiency in Gnutella-like P2P systems. One key question discussed in this thesis is the design of cache in the P2P system. The replacement mechanism for cache design is also briefly mentioned.

3.3.1 What does Cache store?

Basically, there are two types of caches employed by this scheme. One is the request-caching table and the other is the reply-caching table.

RequestID	RequestorID	Keywords	Contenttype
287	A	Yesterday, Carpenters	.mp3
89	B	Love	.mp3

Table 3.1: Example: Peer C's Request-Cache Table

The request-Caching table caches incoming query information. Each query contains a sequence of keywords and specifies what type of files it's looking for. An application program can capture this information from the query and cache it in the request-caching table. To see how this works, let's look at some examples: First, let's suppose that peer C receives a query from A looking for a song with the title "Yesterday once more" by the Carpenters. The query contains the keywords "Yesterday" and "Carpenters" and specifies the content type .mp3. Then suppose peer B is looking for any love songs and peer C gets the query. The second query contains only the keyword "love" and specifies the content type of .mp3. Table 3.1 shows peer C's request caching table after receiving these queries. Of course, there will be a lot of results for B's query, so B may need to refine the query keywords.

An entry is created in the request-cache for each different incoming query. Duplicate queries are not recorded in the cache. Each entry in the cache contains a variable number of keywords, depending on the number of keywords specified in the query, and indicates the type of file sought by the query. In order to save storage space, we could use an intelligent hash function to map the keywords into a series bitmap, as is done in [32].

Reply-Cache table caches all reply messages to the peer's previous queries. Each entry in the reply-cache contains a variable number of keywords, again depending on the number of keywords specified in that query, and indicates the type of file sought by that query. The reply-cache creates an entry for all different replies, not storing duplicate entries. Suppose B queried once with keywords "Love" and content type as music. He got 4 replies: two "Only Love" from different peers, "I just called to say I love you", and "Love Story". They are cached as shown in table 3.2. ReplyID, in the reply-cache table, refers to the initiator of the query and ReplyorID refers to the respondent.

The reply-cache only caches reply-messages that match the peer's queries. The peer may choose the one that it is exactly looking for to download the file from. However,

ReplyID	ReplyorID	Keywords	Content Type
89	C	Love, Only	.mp3
89	D	Love, Called, Say	.mp3
89	E	Love, Story	.midi

Table 3.2: Example: Peer B's Reply-Cache Table

by caching all the reply messages, this peer will grasp more information about the location of similar type of files. And later on it might be able to more quickly resolve similar queries either by directly answering the request or by forwarding the query to the peers in the reply-cache table. A simple algorithm will determine whether two queries match or not. The cache is not intended to cache the entire file. The typical video file contains 100-800 Megabytes. An audio file-.mp3 or wma-contains 3-5 Megabytes. Software file sizes vary widely. But the sizes of the files sought have no bearing on the size of the cache because they are not stored in the cache. Only that information that could be used to expedite similar queries in the future is cached, thus the caching system requires little storage.

3.3.2 Cache hit and Processing algorithm

Since there are two types of caches, there are two kinds of cache hits, each with its own checking priorities. When a node receives a new query, it first checks its own file collection to see if it can satisfy the request. If the node finds that it has the file and can share it, it replies to the request originator and doesn't forward the query to other peers. If the node is not able to satisfy the request from it's own file collection, it check its caches for a similar query. If neither of the caches contains similar queries, the node randomly selects k (which is a system parameter) neighbors to whom it will forward the query. The processing algorithm is shown in table 3.5. It is possible that other solutions would have been identified and cached if the search continued, but when a node has the matching files, it will return just the matching files and the search process via this peer terminates. This method generates far fewer forwarded query messages, thus consuming less bandwidth than the Random Walk method, but returns as many or more results.

When checking reply-cache, the algorithm searches for an exact match. The matching algorithm is quite simple. If an entry contains all the keywords contained in the incoming

ReplyID	ReplyorID	Keywords	Content Type
86	A	k5,k6,k9	software
86	B	K5,k6,k10	software
87	C	K1,k2,k3,k4	music
87	D	K1,k3,k7	music

Table 3.3: An example reply-cache table

query and the cache entry and the query specify the same file type, then there is a hit. The peer in the cache entry is most likely to provide the requested file or to instruct the requesting peer how to get access to the file. For example, suppose a query is initiated, containing keywords “k1” and “k2” and file type “music” when there are 4 entries in this peer’s reply-cache (see table 3.3).

This keyword based search method only returns files that contain all the keywords contained in the query, while search engines return matches that contain any of the keywords. This is because the search in P2P file sharing system is different from the search through search engines. The expected search results or goals are different. In a P2P file sharing system, a peer looks for one or a set of specific files which could be identified through all the query keywords . While searching through web search engines, one may look for all kinds of information associated with the query keywords.

According to the exact-matching algorithm, the third entry is a match. Therefore, this query will be directly forwarded to peer C.

When checking request-cache, a simple “similar-matching” algorithm is used. We say two queries are similar if either of them contains all the keywords of the other and both queries request files with the same content type. Future work may reveal a wiser similar-matching algorithm, but in this thesis the above similar-matching algorithm is accepted for the sake of implementing this caching scheme. If there is a similar query cached in the request-cache table, we call this a hit, which means the peer cached in this entry once requested similar files, and the peer in the cached entry will be more likely than a randomly selected peer, to assist the requestor in accessing the desired file. There is another example. Suppose a query contains the keywords “k1”, “k2” and “k3” and specifies that it is seeking a music file. Currently there are 6 entries in a peer’s request-cache table (see table 3.4).

According to the “similar-matching” algorithm, the third and fourth entries are

RequestID	RequestorID	Keywords	Contenttype
1988	A	k8,k9,k11	movie
1989	A	k1,k4	music
23	B	k2,k3	music
67	C	k1,k2,k3,k5	music
587	G	k6,k7	software
78	F	k1,k2,k3	movie

Table 3.4: An example request-cache table

```
for i : 1..n                    //n is # of shared files
   if q.exact_matching(file[i])
      hits.add(file[i])
if hits != null
   reply(q,hits)
   return
for i : 1..m                    //m is # of records in reply cache
   if q.exact_matching(replycache[i])
      forwardto(replycache[i].peer,q)
      return
count=0
for i : 1..L                    //L is # of records in request cache
   if q.similar_matching(requestcache[i])
      if count ≤K
        forwardto(requestcache[i].peer,q)
        count++
while count<k
   peer p = random_select(neighbor)
   forwardto(p,q)
   count++
```

Table 3.5: Algorithm for processing an incoming query

similar to the query.

Extensions to existing message formats are needed so that keywords and content type items are clearly specified in all query and reply messages in order to fully utilize the capabilities of the caching mechanism discussed in this thesis.

3.3.3 What is the Cache Replacement Strategy?

Hardware storage prices continue to decline. New PCs commonly come equipped with 80-120 Gigabytes of storage on the hard drive. An entry in either Reply-cache or Request-cache takes only a few bytes. Thus, a simple cache replacement strategy (e.g. FIFO, random) might be enough and can be configured at the time of P2P system implementation. When cache is full, the cache entries that are no longer valid will be purged. Only when the cache is full, will the system check which cache entries are no longer valid. The system could give definition on invalid cache entries. For example, the oldest cache entry can be treated as an invalid one.

3.4 Criteria for Performance Evaluation

According to [39], a good search algorithm must be scalable, efficient, and responsive. [39] also proposes a unified search analysis criterion to evaluate the quality of search algorithms in terms of scalability, efficiency, and responsiveness. These criteria are adopted in this thesis and extended to compare the proposed scheme with two popular schemes-Gnutella flooding[2] and Random Walk[16]. This thesis aims to employ the described caching method to improve the search mechanisms of P2P systems and to measure the impact of the caching methods on performance. This caching technique could be combined with other methods to further improve P2P system performance. For example, the method used in [18] caches intermediate node replies. The method proposed in this thesis caches the replies at the request originator. These two methods can be combined so that peers can cache both direct replies and replies that were not directly intended for them. This combination could further improve the hit rate achieved by caching. It's not exclusive. This scheme is compared with two blind search algorithms.

According to [39], a search algorithm is not efficient if it generates a huge number of redundant query messages in an uncontrolled fashion and consequently overwhelms the network's bandwidth. In an efficient search process, there should be a high hit rate. That means that in an efficient search process queries will be highly successful in finding targeted resources.

$$QueryEfficiency = \frac{QueryHits}{\frac{QueryMsg}{NetworkSize}} = \frac{QueryHits}{MsgPerNode} \tag{3.4.1}$$

Search responsiveness includes responsiveness and reliability. Responsiveness is the ability of a search algorithm to respond quickly to meet the needs of a user. Reliability means the ability of a search algorithm to meet the commitments made to users.

$$SearchResponsiveness = \frac{SuccessRate}{HopsNumber} \tag{3.4.2}$$

[39]. In this equation, SuccessRate is the percentage of resolved queries and HopsNumber is the average hop number of the first reply to each query.

In addition, two more criteria are included in the evaluation of the schemes. One is quality of search results and the other is cost associated with the search method.

The satisfaction of search results tell us an important aspect about effective evaluation of a search algorithm in P2P systems. In quality of search results criteria, The following matrix will be used: Average number of results (hits) for each query; average number of results (hits) for each message; the average time to get 1st result; the average hops for 1st reply.

An effective search method shouldn't deplete network resource-CPU processing cycle and storage- in order to find a target. In cost criteria, the following matrix will be used: Average number of messages per node visited; average number of messages generated for each query; space required for caching.

The effectiveness of the caching-based search scheme will be evaluated to determine what percentage of queries are answered through caching and selective-forwarding.

3.5 Summarization

This chapter presents a detailed discussion of the caching based selective query forwarding scheme and its performance evaluation criteria. In chapter 4 simulation design will be discussed in detail.

Chapter 4

Simulation Design

In order to evaluate the three P2P searching schemes, the fundemental method deployed in this thesis is through simulation since it is prohibitively expensive to deploy them in an existing P2P network for evaluation. The simulation design used by M. Schlosser [40] work was modified and extended to resemble real-world P2P networks as closely as possible by referring to the real measurement data [36] in P2P systems.

Basically, the simulation is based on “Query-Cycle” model [40]. The network that is simulated is a Gnutella-like P2P file-sharing network, in which, each peer is interconnected, following a specific distribution. In a query cycle, a peer can actively issue a query for files or respond (resolve, forward or discard) to incoming queries. Each peer, upon receiving an incoming query, can process it. Queries will be delivered and will traverse in the network in some way: broadcast, random walk, or selective-forwarding.

In this chapter, several simulation design issues are highlighted and discussed in the following sections.

4.1 Topology Design

M. Ripeanu et al.[23] ’s work points out Gnutella node connectivity follows a multi-model distribution, and it combines a power law and a quasi-constant distribution. In a typical power law network, most of the nodes have few links (neighboring nodes) and

```
PLOD(N,M,A)=
  FOR i: 1..N
    x = uniform_random(1,N)
    $\text{degree}_i = \beta\ x^{-\alpha}$
  FOR i: 1..M
    while 1
      r = uniform_random(1,N)
      c = uniform_random(1,N)
      if $r \neq c \wedge \text{degree}_r \wedge \text{degree}_c \wedge !A_{r,c}$
        $\text{degree}_r$- -, $\text{degree}_c$- -
        $A_{r,c} = 1$, $A_{c,r} = 1$
        BREAK;
```

(a)Power-Law out degree generator[41]

```
MPLOD(N,A)=
  FOR i: 1..N
    x = uniform_random(1, N)
    $\text{degree}_i = \beta\ x^{-\alpha}$
  FOR i: 1..N
    r = uniform_random(1,N-i)
    if $\text{degree}_i$
        count=0
        while $(\text{degree}_{i+r} \vee A_{i,i+r}) \wedge \text{count} < N-i$
          r++
          if (i+r > N) r=1
        if count $\geq$ N-i
          BREAK;
        else
          $\text{degree}_i$- -, $\text{degree}_{i+r}$- -, $A_{i,i+r}=1$, $A_{i+r,i}=1$
```

(b)Modified Power-Law out degree generator

Table 4.1: Comparison of 2 algorithms generating topology obeying power-law.

very few nodes have large number of links. Specifically, the fraction of nodes with X links, fx, is proportional to X^{-k}. (Here k is a network dependent constant number and also a parameter in the simulation experiments) Although Gnutella is not a pure power law network, it exhibits power-law network characteristics. Hence, the network is initiated into a power-law network.

The PLOD algorithm [41] is referred to generate network topologies that obey power laws. The basic idea of this algorithm is to use the power-law to guide the construction of the undirected graph G(V, E), which is represented in a N $\times$ N adjacency matrix $A_{u,v}$ and in which M edges will be generated. This algorithm manipulates the property of node out-degree. The algorithm in [41] is shown in the table 4.1(a).

The algorithm(table 4.1(b)), which is deployed in the simulation to generate the power-law network, is based on PLOD and is an improvement over PLOD with respect to the running time complexity. For a large power law network generation, PLOD has a very long running time because each time it randomly chooses two nodes which don't have an edge between them, and till most of the edges have been added, it takes an unexpected long time for the algorithm to choose the next available pair of nodes for adding one more edge.

4.2 Content Distribution Model

According to [40], peers in P2P network are generally interested in a subset of the total available content on the network and different type of content has varying popularity. [40] also points out that the popularity of documents in many document storage systems, including the WWW, exhibits Zipf distribution characteristics. That means, some popular documents are held by many peers and thus have a lot of replicas, while most documents are less widely held.

The detailed content distribution model is described from the following aspects:

Assigning Content categories (Refer to [40]) Assume there are n content categories C = c_1, c_2, ..., c_n. Different content types have different popularities. So, they are assigned different popularity ranks. Here, c_1 = 1, c_2 = 2, .., c_n = n, where 1 refers to the highest popularity rank, and n the least popularity rank. Popularity is modeled by a Zipf distribution. That is, when a peer is initialized to be interested in some content type c $\in C$, the probability p(c) is given by $p(c) = \frac{\frac{1}{c}}{\sum_{i=1}^{F} \frac{1}{i}}$. A peer is said to be interested in at least C_{min} content categories. And the exact number of content categories that a peer is interested in, is randomly chosen.

Modeling interest level (Refer to [40]): According to [40], a peer interested in content categories C is probably not equally interested in all categories c $\in$ C and it is more likely to be interested in some categories than others. So, this is modeled by assigning an interest value, w_c^i, which is determined uniformly at random for each content category and each peer i, to each content category c $\in C_i$ that this peer i is interested in. So, the probability that peer i is interested in content category c is given by $p(c|i) = \frac{W_c^i}{\sum_{c' \in C} W_{c'}^i}$. The interest value w_c^i is not supposed to be correlated with the general popularity of content category c, due to the fact that "while a certain category may be of interest to many peers, that category is not necessarily the main interest of those peers"[40]. Further, it is assumed that the network is a steady-state network and the peers don't change its interests over time. (The later assumption is different from the observation made by [37], which concludes that in Kazaa system, where video files are most commonly exchanged and shared, peers tend to change their interests frequently. However, it can be argued that for a relevant short period of time, the assumption is still valid)

Modeling the number of distinct files in each category (Refer to [40] and a little modification): Assume there is maximum replication going on in the network for each content category. And let FC represent the maximum number of distinct files which could be shared by some peer in content category C. Let F represent the total number of distinct files on the network, and p(c) the fraction of files that are in content category c. So, the number of distinct files, F_c, in category c is estimated by $F_c = d \times FC + (1 - d) \times p(c) \times F$. d is some number between 0 and 1 and in the implementation, it is set to 0.25. [40] points out that empirical evidence would be helpful to determine an accurate value for d.

Modeling the number of distinct files in each peer The model of the number of distinct files that each peer is going to share is based on the file distribution measurement done in [36]. A quite obvious percentage of peers are free-riders. Thus, the files that they share are 0. Some peers with high capacity will have a large number of files to share.

Modeling files (Refer to [40] and make some extension): A distinct file held by each peer may be uniquely identified by the tuple {c,r}, in which, c stands for the content category that the file belongs to, and r stands for the file's popularity rank within this content category c. Within each content category, some files are very popular and most are not, so, the file distribution is also modeled as Zipf distribution. When a file is generated in a peer i, the probability that this file is in category c is given by $p(c|i)$ which has been given in the above section, and the probability that this file has a popularity rank r is given by $p(f_{c,r}|r) = \frac{\frac{1}{r}}{\sum_{i=1}^{F} \frac{1}{i}}$.

In addition to the content category and popularity, a file also has a list of keywords. Different files may have different number of keywords. The number of keywords that consist of a file is determined uniformly at random, ranging from 1 to MaxNumberofKeywords. Each keyword is a random integer.

4.3 Network Delay

A model in which the delay for a query being transferred from one peer to another peer on an overlay network is chosen randomly is not accurate, as it will fail to identify the physical position of these peers in an overlay network. Two peers, which are neighbors with each other in an overlay network, may happen to be far apart from each other in the physical IP network. To accommodate the physical consideration, the network delay is modeled into three classes.

There are three classes of delay: delay between two peers from same AS; delay between two peers from two ASs, which are in the same continent; and delay between two peers from two ASs, which are in different continent. The network delay value for each class has different order of magnitude. Moreover, the delay value between two peers is variable. It is generated each time a message is transferred between two peers.

4.4 Query Generation

No real measurement of this query pattern for P2P system could be found yet. Thus the query generation is modeled based on common sense.

Each peer can issue queries. The query is generated based on Poisson process. Each time an active peer in the network is randomly selected as query generator. Each selected peer will generate a query according to its interests, which have been set at the beginning of network construction and initialization.

A query consists of a tuple {c, a list of keywords }, in which, c represents the content category c, and the list of keywords represent the keywords that the desired file is expected to have.

The list of query keywords are not randomly generated since, otherwise, we lose the meaning of file popularity. The query keywords are chosen from the keywords of a file with content category c and popularity rank r. The number of query keywords is determined uniformly at random from 1 to the number of keywords of the file f(c, r). Then relative numbers of query keywords are randomly chosen from the keywords list in the file f(c, r). The probability that file f(c, r) is chosen for the query generated by peer i, is given by $p(f_{c,r}|i) = p(c|i) \times p(f_{c,r}|c)$

4.5 Network Dynamic Property

It won't be an accurate model if peer dynamic behavior is ignored. In fact, peers' dynamic behavior could have a great impact on a network and the performance of search algorithm. For example, if nodes frequently join and leave a Chord network, the administration overhead will be very huge.

To respect the dynamic property, each peer can be in two states: active/inactive. Each peer is dynamically re-joining and leaving the system, which means it can or cannot generate queries or respond to incoming queries.

The probability of a peer being on-line is assigned according to the uptime distribution in [36]. The session duration is based on exponential distribution. Distribution and the average session duration time is based on the measurement value in [36].

4.6 Peer Modeling

Each peer maintains a neighbor list, reply-cache list, request-cache list, AS number, state (Active or in-Active), file collection that it would like to share with other peers and its interest list.

Each peer has different interests for different content category. This is not co-related to the popularity of content category. The number of files shared in each peer is generated according to the distribution measured in [36]. All peers would not like to equally share the files, and some low-capacity peers may prefer not to share files. For such peers, the number of files shared is 0. This has been covered in detail in the previous sections.

4.7 Implementation

C++ language is used to implement the simulation since C++ is a good Object-oriented language, which is more suitable for this project. There are eight major classes: CPeer, CNetwork, CExperiment, CRequest, CReply, CEntry, CEventq, CFile.

Since network is a combination of many queuing systems, and hence a simulation experiment is based on an event queue. A global event queue is maintained for event scheduling. The event types are: Peer Joining, Peer Leaving, Peer Receiving a Request, Generate a new request, Receive Request, Receive Reply. (Overall there are six types). There are three basic operations for queue management: queue initialization, insert event into and get event from a queue. The event queue is an ordered linked-list according to the scheduled time of the event.

Chapter 5

Performance Evaluation

This chapter presents the performance metrics in section 5.1. Section 5.2 discusses the parameters used in the simulation experiments and justifies the choice of some of the parameter values. The simulation experiment results are shown and discussed in section 5.3.

5.1 Metrics Used

According to the criteria of evaluating the performance of search schemes described in section 3.4 of chapter 3, the following metrics will be adopted to evaluate the performance of scheme proposed in this thesis, flooding[2] and Random Walk [16]:

- **Query Efficiency (QE)**: $= \frac{QueryHits}{QueryMessage/NetworkSize} = \frac{QueryHits}{MsgPerNode}$. It depicts how successfully the query messages generated during the search process find the target objects.

- **Search Responsiveness (SR)**: $= \frac{SuccessRate}{HopNumber}$. Here success rate means the percentage of queries with hits. HopsNumber means the average hops number for the first reply to each resolved query. Search Responsiveness represents the ability of a search algorithm to respond quickly to meet the needs of a user and the commitments made to users.

- **Quality of Search Result in the following aspects**:
  - Msg/Query: it refers to how many query messages a query will generate? It depicts how much traffic a query will cause in a network. Reducing Msg/Query will increase the scalability of the Gnutella-like P2P systems
  - Hits/Query: it refers to how many results (exact matching for a query) a query will get? It depicts how successfully a query is resolved. A valid search algorithm should not sacrifice the Hits/Query too much to achieve a little improvement in scalability.
  - Hits/Msg: it refers to how many results (exact matching or hit for a query message) a query message will get. It somehow depicts the efficiency of a message.
  - First Reply response time(1st Resp. time): it refers to the average time for a peer to get the first reply for its query. It depicts how fast a query will get answered.
  - Average 1st reply pathlen: it refers to the average hop length for the first reply that a peer gets for its query. It depicts how long it takes for the first reply to come back.
- **Caching Hit Rate(CHR)**: It refers to the percentage of queries resolved through caching. If CHR is very high, then caching has the potential to improve performance of P2P systems.
- **Average Storage for caching per peer**: This is measured in terms of number of entries in reply-caches and request-caches in each peer. An algorithm should not deplete the whole system's storage to achieve a little improvement in scalability. Please note that peers who have just joined the system do not have any entries in caches, and thus have no choice but to randomly choose several neighbors to forward the query and locate the files.

5.2 Parameters and Justification

During the simulation experiments, the parameters are shown in table 5.1. It is impractical to vary all of the parameters to evaluate the performance of the proposed

Parameter Name	**Value**	
Peer Number(N)	2000	
AS number	10	
Continent number	3. (Each continent has 6,3,1 ASs accordingly)	
Latency between two peers	Within same AS	70ms ˜ 80ms
	Within Same Continent	140ms ˜ 160ms
	Within Different Continent	280ms ˜ 330ms
Maximum Number of File(F)	10000	
Maximum Number of File could be shared by some peer in content category C(FC)	100	
CONTENT Types	10	
Maximum file popularity	Depends on the total number of files in each content type.	
Maximum # of interest types per peer(C_{max})	10	
Minimum # of interested types per peer(C_{min})	3	
MaxNumberofKeywords	4	
Average Query Generation Rate	1 query per minute	
Simulation time	1200 hours	
TTL	Vary in simulation	
Average Degree for each peer and K walker	Vary in simulation	

Table 5.1: Parameters Table

scheme with the other two schemes: flooding [2] and random walk [16]. Instead, several important parameters are picked to vary while other parameters are kept constant.

A popular peer-to-peer network can have as many as 20,000 nodes [2]. However due to the computational power of available PCs, the simulated network size is set to 2000 as a representative. The size of network won't have much effect on the conclusion that we will get, but only reduce the time for completing a simulation experiment. (The finishing time of a simulation experiment on a flooding based Gnutella network will grow dramatically as the network size increases). There are 10 ASs, each with an average 200 of peers as a representative. There are 7 continents in the world, with 3 continents (North America, European, Asia) having most ASs. So, the continent number is set to 3. Among the 3 continents, North America has the biggest percentage of ASs.

The latency between any two peers is referred to [36]. It says, the closest peers are four times closer than the furthest peers. The furthest peers have latencies of at least 280ms, and the closest peers have latencies of about 70ms.

According to the measurement value in [2], 25% of the Gnutella clients don't share any files, 30% of the Gnutella clients share less than 10. So average number of files for each peer is set to 5 as a representative. Hence, the maximum number of files (F) is set to 2000 $\times$ 5 = 10000. Maximum Number of Files shared by some peers in content category C(FC) is set to 1% of 10000, which equals to 100. Empirical evidence would be useful to determine an accurate value of the percentage.

Although in Gnutella system, any type of files could be exchanged, according to [20], around 27% of downloaded files in P2P system are of music type; around 45% are of movie type; and 28% are of application type. Without loss of generality, the available number of content categories is set to 10 (which is equal to the maximum number of interested types per peer), with each having different popularity rank and different amount of files. "Music, movie, application" amount to 90% of exchanged files in P2P systems. Thus the minimum number of interested types per peer is set to 3.

In the implementation, MaxNumberofKeywords in a file or a query is set to 4. Also empirical evidence would be helpful to determine an accurate choice of MaxNumberofKeywords.

There are three classes of time granularity. Latency between two peers has time granularity measured in "microsecond(ms)", the query generation rate has time granularity in "minute", and the simulation time has time granularity in "days or months". With these consideration, the query generation rate is set to 1 per minute, and the simulation is set to 1200 hours.

The performance of Gnutella flooding, Random Walk, Sgnutella(the scheme proposed in this thesis) are compared by varying the TTL and average degree (the average number of neighbors) of each peer, and all the other parameters are kept constant and the same for the three schemes.

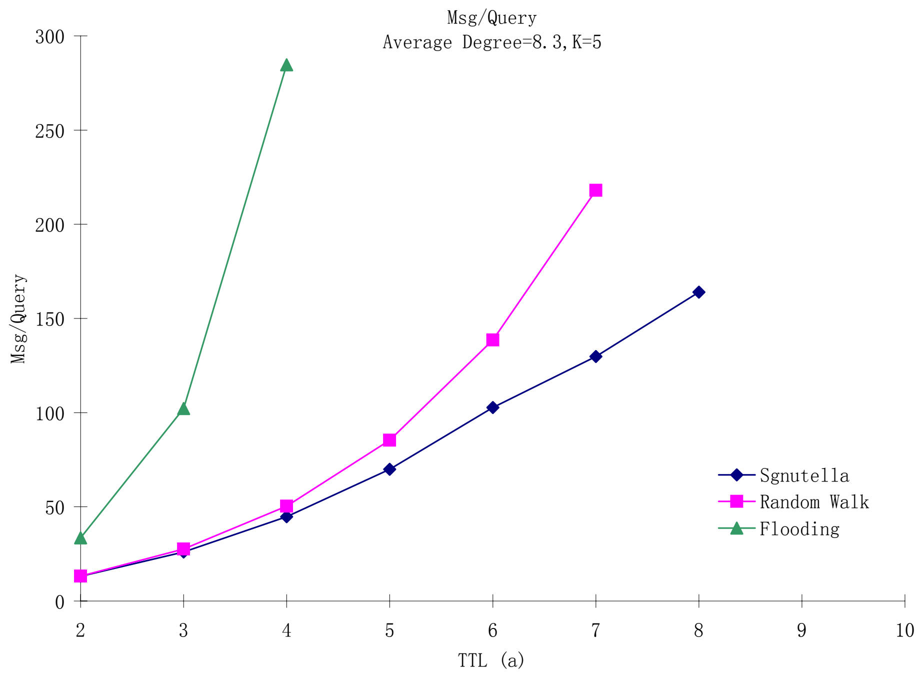
5.3 Results and Analysis

In the following figures, Sgnutella stands for the caching and selective query-forwarding algorithm, flooding stands for the Gnutella flooding algorithm.

As we can see and expect from figure 5.1(a) and (b), in either case, flooding generates the largest amount of query messages for each query and the scheme with caching and selective query forwarding generates the fewest. The number of messages generated for a single query through flooding is proportional to d^{TTL}, in which d is the average out links number of a peer. With the increase of TTL, the number of messages generated for each query through flooding grows exponentially. Thus Gnutella flooding network has poor scalability. As we can see from the figures, Gnutella data stops being collected after TTL=4. This is becasue under this scenario where average out link numbers of each peer is 8.3, and TTL ≥ 5, each query generates more than 8.3^5 query messages. It takes several days or longer to finish such a simulation experiment. Random walk greatly reduces the unnecessary query messages, at the expense of reduced hit rate of finding targets for each query. Caching and selective query forwarding, through learning from past experience, further reduces unnecessary query messages.

As we can see and expect from figure 5.2, Gnutella flooding algorithm generates the highest hit rate for each query, due to its aggressive flooding nature, and the scheme with caching and selective query forwarding comes next due to its learning nature. Random walk achieves the lowest hits per query due to its blind searching nature. From figure 5.1 and 5.2, we can see that the topology of the network has significant effect on the performance of the schemes. The problem of how well the search algorithms could be able to adapt to the topology needs further investigation.

As we can see from figure 5.3(a) and (b), in either case, Query Efficiency of Gnutella flooding algorithm falls below that of Random Walk algorithm, which also falls below that of Sgnutella algorithm. Although Gnutella flooding algorithm has the highest hit rate of finding an object target, because of the flooding nature, it also generates much larger amount of messages for a single query, and thus counteracts that effect and even leads to the lowest query efficiency among the three algorithms. As seen from figure 5.1 and 5.2, random walk algorithm has a lower hit rate of finding an object target, while it generates larger amount of messages for a single query, thus it unavoidably has a lower QE. This is due to blind searching nature of random walk. It doesn't learn from the past experience



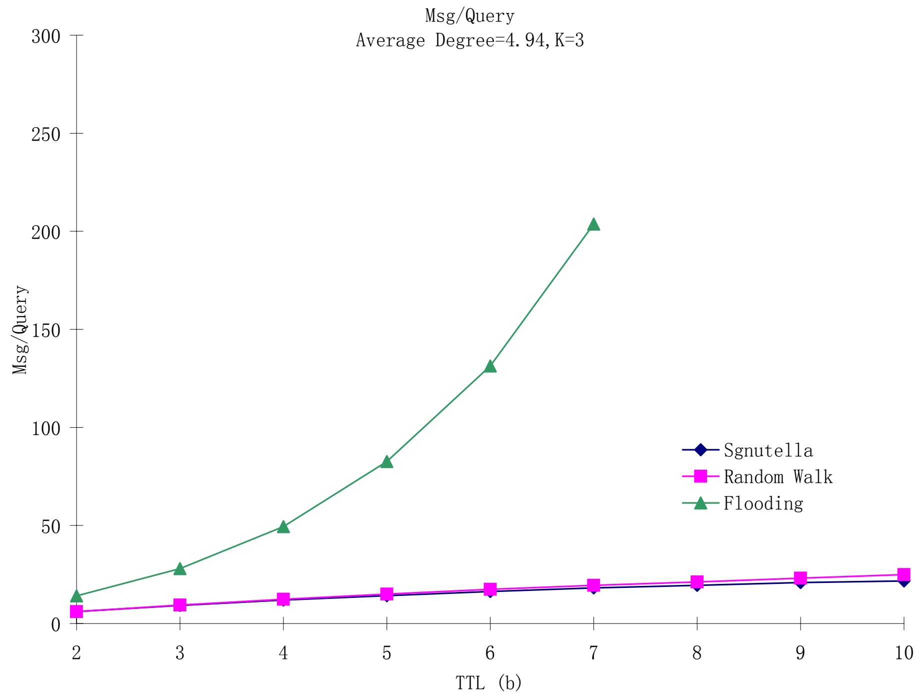
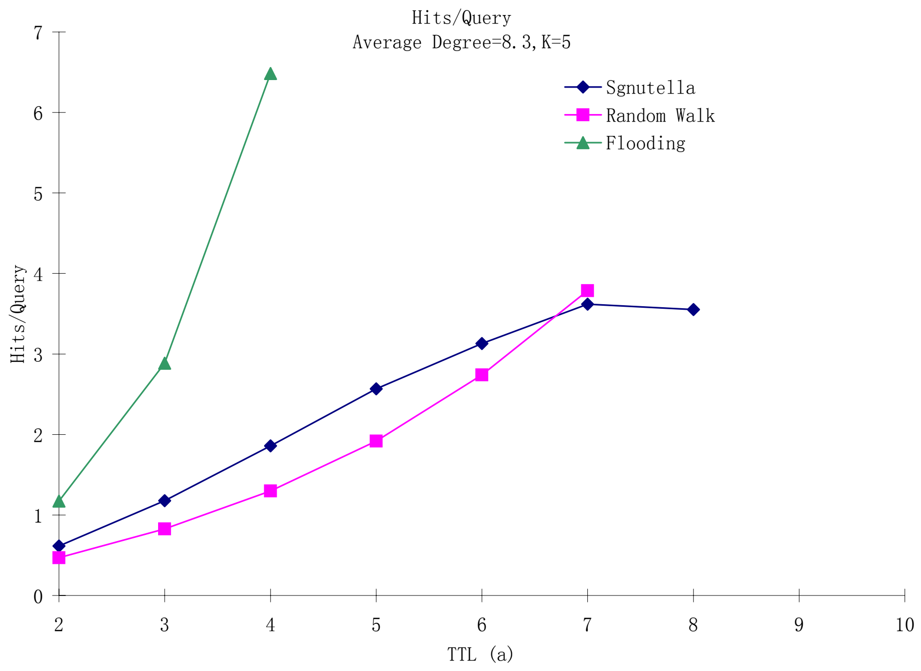


Figure 5.1: Msg/Query comparison of 3 algorithms simulated in Gnutella Network



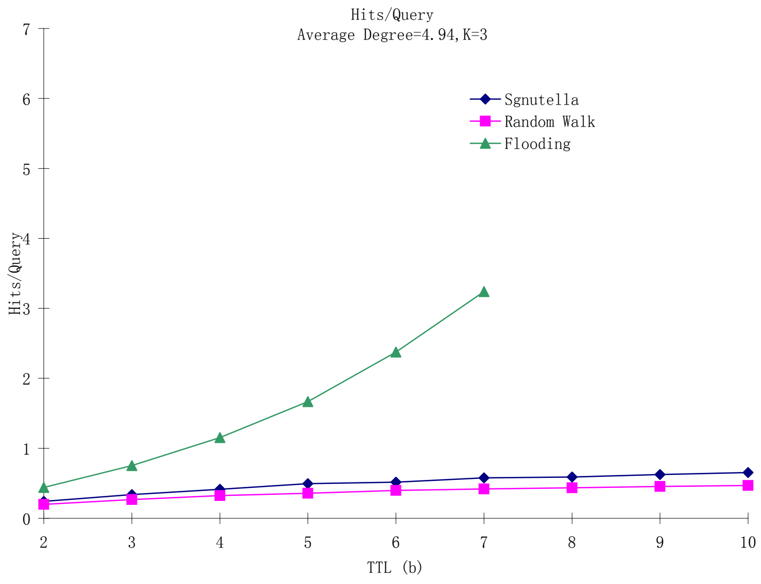
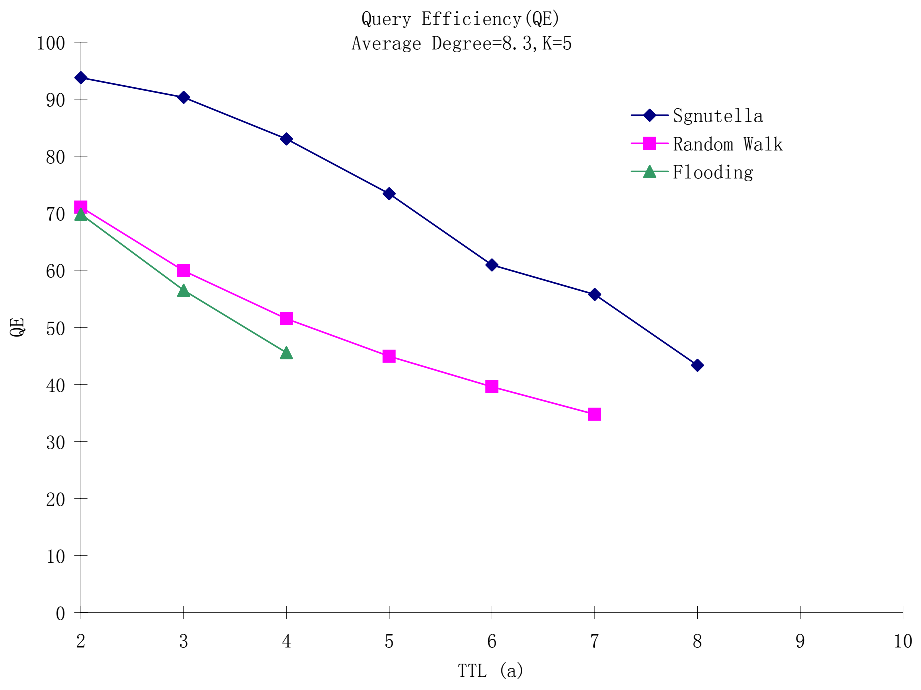


Figure 5.2: Hits/Query comparison of 3 algorithms simulated in Gnutella Network



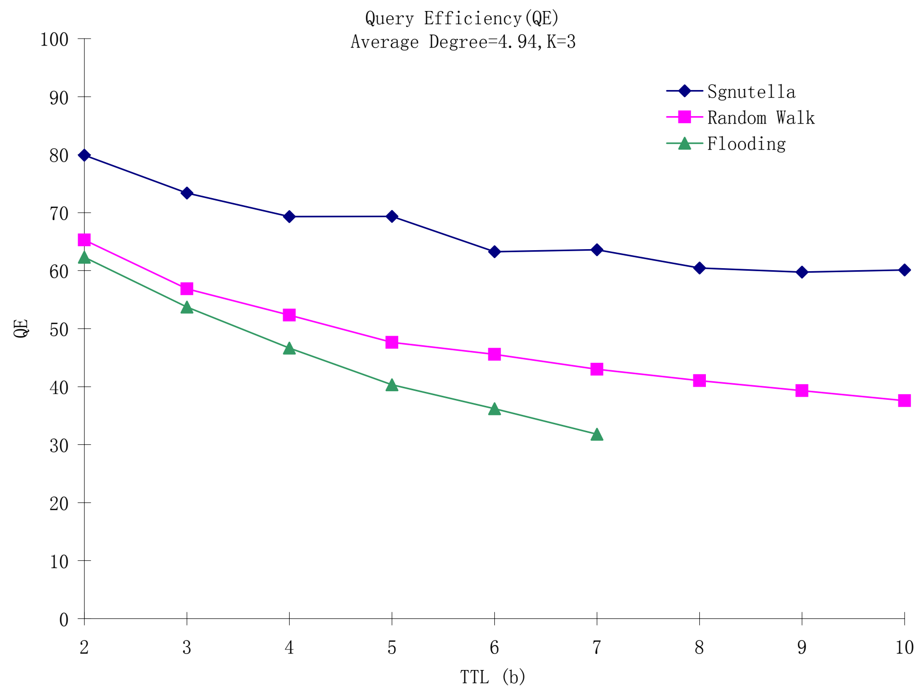
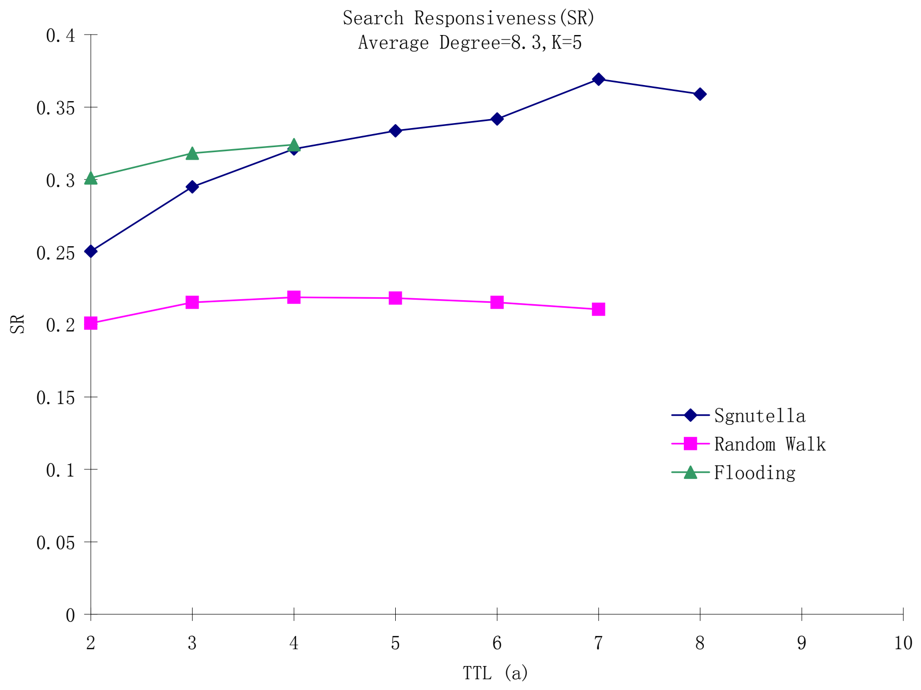


Figure 5.3: Query Efficiency comparison of 3 algorithms simulated in Gnutella Network

and have the knowledge of which neighbors are most likely to answer a query. The caching and selective query forwarding algorithm learns from the past queries and replies. When choosing neighbors to forward the query, it has a better knowledge of who will be most likely to resolve the query. Compared with flooding algorithm, it sacrifices the hit rate of finding an object target, it reduces unnecessary query message forwarding. Compared with random walk algorithm, it uses the knowledge learned from the past, and thus has much higher hit rate of finding a target. Thus it has the highest QE. The query efficiencies of the three algorithms will decay with respect to the search hops as the number of query messages increases much faster than that of the number of query hits.

As we can see from figure 5.4 (a) and (b), in either case, Gnutella flooding algorithm has highest Search Responsiveness, caching and selective query-forwarding algorithm ranks the second, and random walk ranks the lowest. Not surprisingly, flooding algorithm has the highest Search Responsiveness, since each time a peer aggressively sends the query messages by forwarding the query message to all of its neighbors. It has the highest success rate of finding a specific target. As we can see from figure 5.2, Gnutella flooding algorithm has far higher hit rate than the others. Caching and selective query-forwarding algorithm has the second highest success rate of finding a specific target. As what has already been explained, caching and selective query-forwarding algorithm has overcome the blinding search nature of random walk, it builds knowledge over past experience and can select the neighbors which are most likely to resolve the query, thus it has higher success rate of finding a specific target and generates faster response than random walk.

Figure 5.5(a) and (b) show different results. It seems “strange” in (b) that Gnutella flooding has the highest average first reply response time, which is expected to be the lowest or lower than that of Random Walk. The difference between the two figures lies in the difference of the underline network. In (b), the network is configured that the average out links number for each peer is 4.9, which is much less than 8.3 in (a). As seen from figure 5.2 (b), the hits per query for the three algorithms are all less than that in (a), which tells us that the average out links number has significant effect on finding the targets. In the simulated network in (b), Random Walk algorithm produces very low hit rate for each query than that of Gnutella flooding algorithm. So, in the computing of average response time for the first reply, more results which are founded within further scope in flooding algorithm are included in computation than that of Random Walk. The proposed scheme has a better response time.



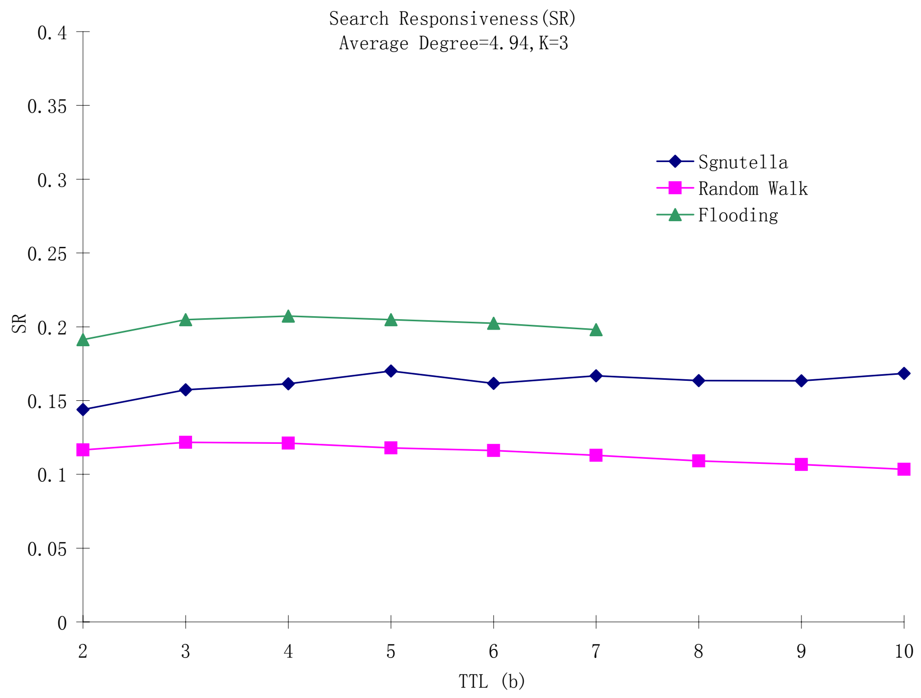
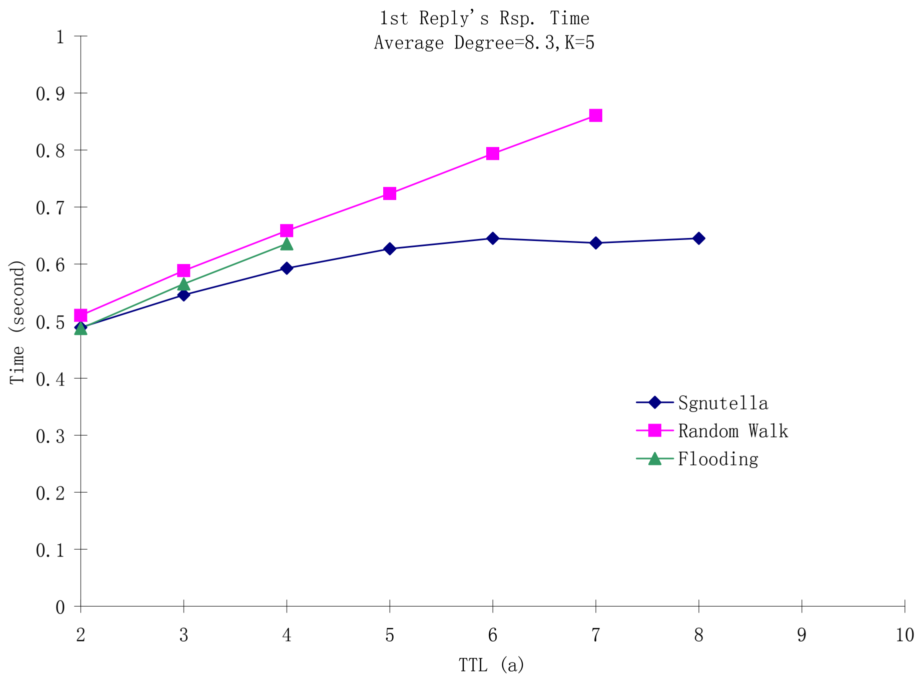


Figure 5.4: Search Responsiveness comparison of 3 algorithms simulated in Gnutella Network



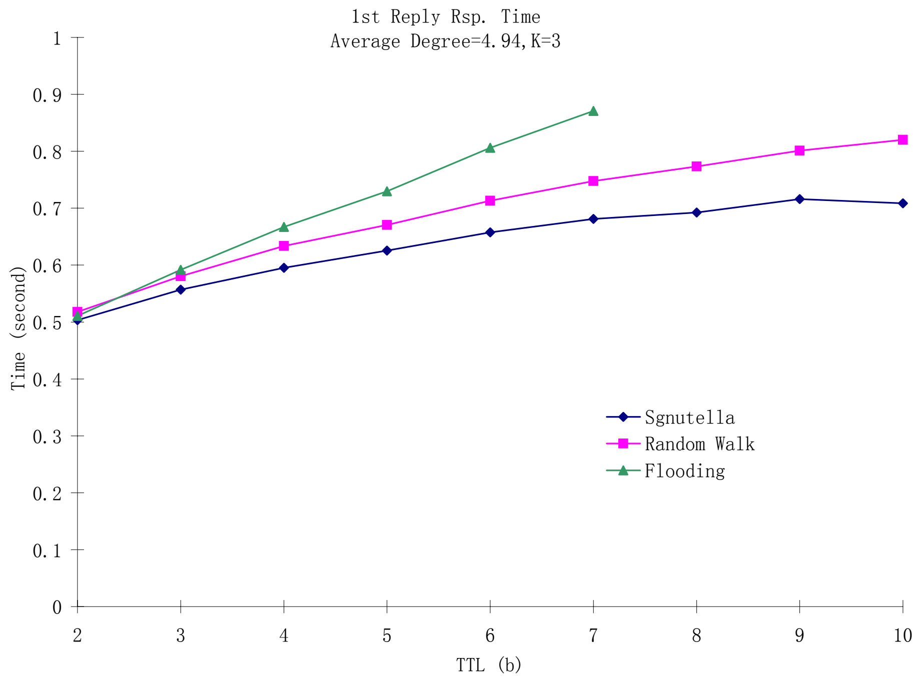
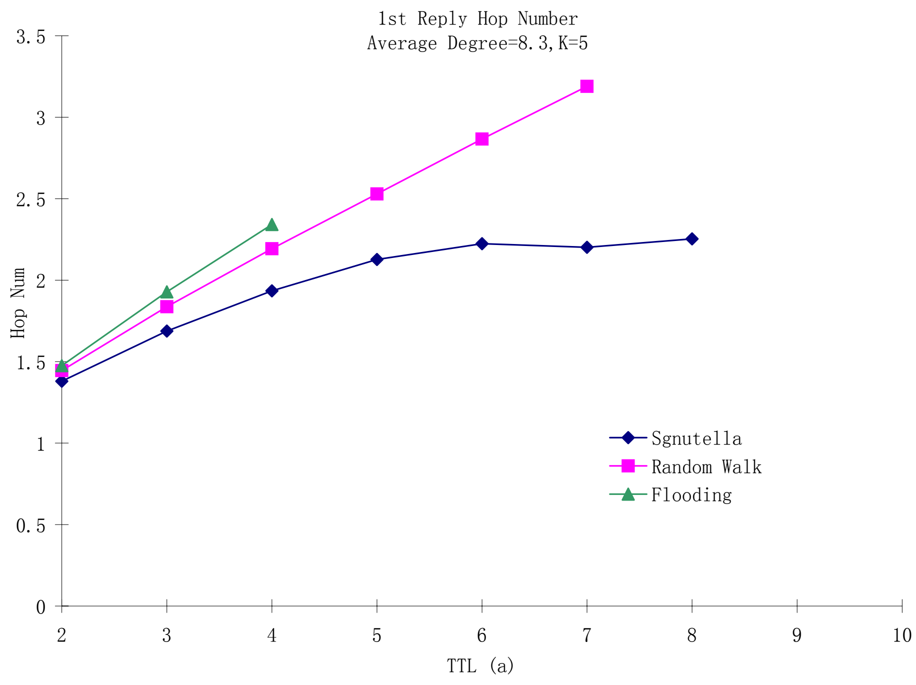


Figure 5.5: First Reply's Response Time comparison of 3 algorithms simulated in Gnutella Network



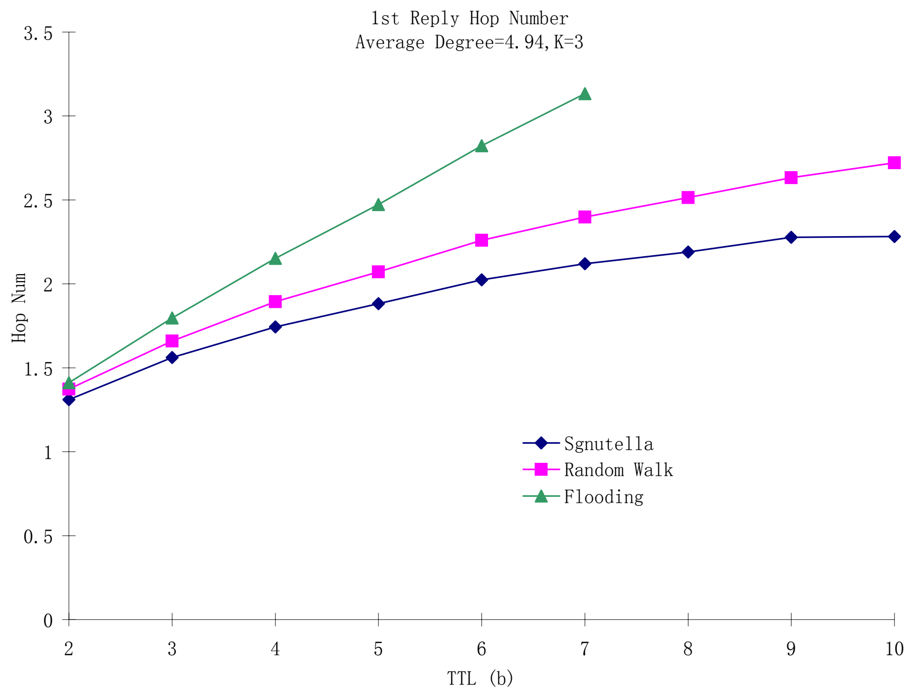


Figure 5.6: First Reply's Hop Number comparison of 3 algorithms simulated in Gnutella Network

	Hop=1	Hop=2	Hop=3	Hop=4	Hop=5	Hop=6	Hop=7	Hop=8	total
TTL=2	0.277	0.204							0.481
TTL=3	0.226	0.169	0.200						0.595
TTL=4	0.184	0.143	0.163	0.185					0.675
TTL=5	0.157	0.126	0.137	0.147	0.159				0.726
TTL=6	0.143	0.117	0.121	0.122	0.122	0.121			0.746
TTL=7	0.139	0.144	0.121	0.107	0.093	0.084	0.078		0.746
TTL=8	0.147	0.137	0.106	0.092	0.081	0.069	0.061	0.0544	0.748

Table 5.2: Caching Hit Rate, in terms of percentage of results resolved through caching. This is measured when the average degree is 8.3 and K is 5

As seen from figure 5.6, the scheme with caching and selective query-forwarding has lowest hop length for the first reply which tallies with the result in figure 5.5. However, Gnutella flooding scheme has the longest hop length for the first reply. Gnutella-like P2P systems are overlay networks, which are built upon IP network. Two neighboring peers may even have larger network latency than that of two peers with several hops apart. Thus, smaller response time doesn't guarantee smaller hop number.

From the above figures, we can see that the structure of network has great effect on the performance of the algorithms. Average out links number (out degree) of each peer determines the density of network connectivity. As we can easily see from figure 5.1 and 5.2, the denser (the higher the average out degree) the network is, the larger the amount of generated query messages for a single query is, and the higher the hit rates are and the lower TTL an algorithm can support.

Table 5.2 gives us a general picture of what hit rate caching mechanism could achieve. That is to say, how many percentages of results could be resolved through caching. The hit rate increases as TTL increases since a query could be cached by more peers, and around 75% resolved queries could be through caching. Even at the lowest TTL setting, 48% hits are through caching. This result confirms that caching is a potential technique to improve the systems' performance.

Table 5.3 gives us a general sense of how much "cost" are needed for caching. As been talked about in chapter 3, a request-cache entry includes "request ID", "requestor ID", "keywords of the query", and "content category for the desired files". A reply-cache entry includes "reply ID", "replier ID", "keywords of the file", and "content category of the replied file". In both cases, a cache entry won't exceed 50Bytes. After 72300 queries

	# of entries in request-cache	# of entries in reply-cache
TTL=2	188.543	2.506
TTL=3	350.538	3.821
TTL=4	586.163	4.635
TTL=5	883.537	5.0955
TTL=6	1225.68	5.5005
TTL=7	1446.28	6.072
TTL=8	1726.88	6.015

Table 5.3: Average number of entries cached in each peer when 72300 queries are issued.

issued in a network of size 2000, within 1200 hours (please refer to the parameter in section 5.2), even at TTL=8, the cost won't exceed (1726.88+6.015)$\times$50Bytes =87Kbytes. Even at higher rate of generating queries, the cost will fall in the scope of several Mbytes, which is far smaller than the huge hardware storage. In real practice, we can set a cache size. When the cache is full, a cache replacement algorithm can be deployed. It's reasonable to conclude that the caching scheme doesn't need to consume a large storage, and this method is practical.

Finally, three approaches are considered when there is a need to forward a query. When there are several peers found in the reply-cache, who have files that exactly match the query, which one should we select to forward the query to? The three approaches are: select the first K peers found; select the peer, which is possibly the closest to the query requestor; select the peer, which made a most recent reply to this peer. In a highly dynamic network, a peer may not be available any more when a query is forwarded to it by caching and selective query forwarding. So, selecting K peers instead of only one to forward the query will possibly improve the chance of hits greatly. However this method may incur more messages. The approach that selects the peer who is possibly the closest to the query requestor, will possibly generate the quickest response and fewer messages than previous one. The approach that selects the peer, which made the most recent reply, may generate higher hit rate than the second one, since the peer is more likely available. Experimental results show that selecting K peers each time does a better performance than that with only one peer selected each time with respect to the performance evaluation criteria. The figures are not shown in this thesis paper in order to save space.

Chapter 6

Conclusions and Future work

6.1 Conclusions

Peer-to-peer (P2P) applications have recently gained popularity these years. P2P networks allow hosts to share their resources in a distributed manner. Among P2P systems, Gnutella-like P2P systems offer several advantages in simplicity of use, self-organization, robustness, fault-tolerance and no need of complex structure control. However the flooding based search algorithm makes Gnutella-like systems inefficient. Thus Gnutella-like P2P systems suffer from poor scalability and are prone to cause great traffic. Much work has been done to address the problem of efficient information retrieval in such systems. Caching is an effective way of improving the performance of any system that makes repetitive requests. However little work has been done to investigate how caching in P2P search mechanism could improve information retrieval. This thesis proposes a selective query-forwarding scheme based on caching to enhance efficiency of information retrieval in P2P systems. This is a promising approach to introduce performance enhancements to information retrieval. The performance of the scheme proposed in this thesis is evaluated and compared with two other well-known techniques in Gnutella-like systems: flooding and random walk. The simulation results show this method offers great improvements in efficiency of information retrieval.

This caching and selective query forwarding scheme makes use of the query locality principle. That is two queries looking for the same file will probably have similarities, which can be reflected through the keywords used in the queries. They will probably have common keywords. Through checking the similarity between incoming query and cached queries or replies, messages can be forwarded directly to those peers who are most able to resolve the query quickly.

The results indicate that the caching and selective query forwarding scheme is an attractive and effective technique for keyword searching in Peer-to-Peer systems. This approach is fully distributed and scales well with the size of network, since it doesn't incur additional messages to maintain or exchange file location information. Compared with flooding, although it has lower search responsiveness, the scheme has produced many benefits in other performance aspects. It has higher query efficiency than that of the flooding and random walk methods. It generates fewer query messages than the flooding and random walk methods. When multiple cached queries or replies are found, multiple peers may be selected to receive the forwarded queries, thus improving the chances that the target objects will be found quickly and reducing the cost of finding it. The caching and selective query forwarding scheme overcomes the shortcoming of pure flooding which always make visits to those peers that don't have desired objects and generates large amount of unnecessary query messages. It also overcomes the shortcoming of pure random walk method, which does blind search. The results also show this approach requires smallest query response time to get the first reply and the smallest hops to reach the first hit. In addition, caching mechanism shows a very high hit rate since as high as 75% query hits could be achieved through caching. Lastly, the proposed approach doesn't require much storage to deploy; this merit makes it practical in reality.

6.2 Future Work

In the future, I plan to continue investigating the proposed search algorithms and do more performance evaluations. Collecting actual traces of search terms by users will be critical to evaluate this cache-based scheme and to develop a more accurate model. Areas that I intend to further explore include the development of more accurate keywords generation pattern, the dynamic popularity of files, and dynamic generation of files. For the further performance evaluation, the percentage of matching files actually returned to requestor will be added into the simulation metrics. I also plan to further investigate other methods of improving the cache hit rate, e.g. either by changing the way that cache is updated and by distributing information differently throughout the network. I further plan to revise the matching algorithm. Currently, the matching algorithms only return "yes/no" answers for either query-query match checking or query-reply match checking. In future work, I will revise the matching algorithms to return an index, through which, it will be possible to measure the degree of similarities. By using an index of similarity, it may be possible to make wiser decisions about query message forwarding. I am also interested in digging into keyword search issues and their effect on the performance of caching. For selective query forwarding, more heuristics rules could be investigated, e.g. bandwidth awareness forwarding, and load-balancing. Finally, I intend to investigate and discuss how to engage a node that is protected by a firewall and how to handle the loss of some nodes from the network while they are still referenced in the cache.

Bibliography

[1] Napster website: http://www.napster.com.

[2] Gnutella website: http://gnutella.wego.com.

[3] 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 ACM SIGCOMM '01 Conference*, San Diego, California, USA, August 2001.

[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker: A scalable Content-Addressable Network. In *Proceedings of the ACM SIGCOMM 2001*, San Diego, California, USA, August 2001.

[5] A. Rowstron and P. Druschel: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, November 2001.

[6] Kazaa website: http://www.kazaa.com.

[7] The practice of peer-to-peer computing: Introduction and history http://www-106.ibm.com/developerworks/java/library/j-p2p.

[8] Seti@Home website. Http://setiathome.ssl.berkeley.edu.

[9] A. Rowstron and P. Druschel: Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the*

eighteenth ACM symposium on Operating systems principles, Banff, Alberta, Canada, October 2001.

[10] N. Daswani, H. Garcia-Molina and B.Yang: Open Problems in Data-Sharing Peer-to-Peer Systems. In *Proceedings of the 9th International Conference on Database Theory (ICDT'03)*, Siena, Italy, January 2003.

[11] Groove Networks website. http://www.groove.net.

[12] Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker and Sonesh Surana: Internet Indirection Infrastructure. In *Proceedings of ACM SIGCOMM 2002*, Pittsburgh, PA, USA, August 2002.

[13] openP2P website: http://www.openp2p.com.

[14] Internet2 P2P working group http://p2p.internet2.edu.

[15] The impact of file sharing on service provider networks. An industry white paper, Sandvine Inc.

[16] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker: Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of the 16th international conference on Supercomputing (ICS'02)*, New York, NY, USA June 2002.

[17] D. Tsoumakos and N. Roussopoulos: Adaptive Probabilistic Search (APS) for Peer-to-Peer Networks. In *Proceedings of the 3rd IEEE International Conference on Peer-to-Peer Computing (P2P'03)*, Linköping, Sweden, September 2003.

[18] V. Kalogeraki, D. Gunopulos and D. Zeinalipour-Yazti: A local search mechanism for peer-to-peer networks. In *Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM'02)*, McLean, Virginia, USA, November 2002.

[19] J. Chu, K. Labonte and B. N. Levine: Availability and locality Measurements of Peer-to-Peer File Systems. In *Proceedings of ITCom: Scalability and Traffic Control in IP Networks*, Boston, Massachusetts, USA, July 2002.

[20] N. Leibowitz, A. Bergman, R. Ben-Shaul, and A. Shavit: Are file swapping networks cacheable? Characterizing P2P traffic. In *Proceedings of the 7th International Workshop on Web Content Caching and Distribution (WCW)*, Boulder, Colorado, USA, August 2002.

[21] L. Fan, P. Cao, J. Almeida, and A. Broder: Summary Caches: A Scalable Wide-Area Web Cache Sharing Protocol. In *Proceedings of ACM SIGCOMM Conference*, Vancouver, Canada, September 1998.

[22] D. L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker: Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proceedings of IEEE INFOCOM 1999*, New York, NY, USA, March 1999.

[23] M. Ripeanu and I. Foster: Mapping the Gnutella Network - Macroscopic Properties of Large-scale P2P Networks. *IEEE Internet Computing Journal*, 2002.

[24] Y. Xie and D. O'Hallaron: Locality in Search Engine Queries and Its Implications for Caching. In Proceedings of IEEE INFOCOM 2002, New York, NY, USA, June 2002.

[25] P. Ganesan, Q. Sun and H. Garcia-Molina: Yappers: A Peer-to-Peer Lookup Service over Arbitrary Topology. In *Proceedings of IEEE INFOCOM*, San Francisco,California, USA, April 2003.

[26] K. Sripanidkulchai, B. Maggs, and H. Zhang: Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. In *Proceedings of IEEE INFOCOM*, San Francisco, California, USA, April 2003.

[27] K. Nakauchi, Y. Ishikawa, H. Morikawa, and T. Aoyama: Peer-to-peer keyword search using keyword relationship. In *Proceedings of 3rd International Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems (GP2PC 2003)*, Tokyo, Japan, May 2003.

[28] The Gnutella Protocol Specification v0.4, 2000. http://www9.limeware.com/developer/gnutella_protocol_0.4.pdf.

[29] FastTrack website. Http://www.fasttrack.nu.

[30] D. Tsoumakos and N. Roussopoulos: A comparison of Peer-to-Peer Search Methods. In *Proceedings of the Sixth International Workshop on the Web and Databases*, San Diego, USA, June 2003.

[31] Z. Xu and Y. Hu: SBARC: A Supernode Based Peer-to-Peer File Sharing System. In *the 8th IEEE Symposium on Computers and Communications (ISCC'03)*, Kemer-Antalya, Turkey, June 2003.

[32] Query Routing for the Gnutella Network. http://www.limewire.com/ developer/query_routing /keyword%20routing.htm#_edn3.

[33] M.S. Khambatti, K.D. Ryu and P. Dasgupta: Push-Pull Gossiping for Information Sharing in Peer-to-Peer Communities. In *Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Las Vegas, Nevada, USA, June 2003.

[34] A. Crespo and H. Garcia-Molina: Routing Indices for Peer-to-Peer Systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, Vienna, Austria, July 2002.

[35] B. Yang and H. Garcia-Molina: Improving Search in Peer-to-Peer Networks. In *Proceedings of 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, Vienna, Austria, July 2002.

[36] S. Saroiu, P. Gummadi, and S. Gribble: A measurement Study of Peer-to-Peer File Sharing Systems. In *Proceedings of the Multimedia Computing and Networking (MMCN)*, San Jose, California, USA, January 2002.

[37] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy and J. Zahorjan: Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP'03)*, Bolton Landing, New York, USA, October 2003.

[38] S. Sen and J. Wang : Analyzing peer-to-peer traffic across large networks. In *Proceedings of 2nd Annual ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

[39] T. Lin and H. Wang: Search Performance Analysis in Peer-to-Peer Networks. In Proceedings of the 3rd IEEE International Conference on Peer-to-Peer Computing (P2P'03), Linköping, Sweden, September 2003.

[40] Mario T. Schlosser, Tyson E. Condie, and Sepandar D. Kamvar: Simulating a File-Sharing P2P Network. *1st Workshop on Semantics in P2P and Grid Computing*, Budapest, Hungary, December 2002.

[41] C. R. Palmer and J. G. Steffan: Generating Network Topologies That Obey Power Laws. In *Proceedings of the Global Internet Symposium, Globecom 2000*, San Francisco, California, USA, November 2000.

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