This is to certify that the
dissertation entitled

DIAS: DISTRIBUTED INTEREST-BASED ADAPTIVE SEARCH FOR P2P NETWORKS

presented by

Ning Liu

has been accepted towards fulfillment
of the requirements for the

Master degree in Department of Computer Science and Engineering

Li Xiao
Major Professor's Signature

8/7/2003
Date

DIAS: DISTRIBUTED INTEREST-BASED ADAPTIVE SEARCH FOR P2P NETWORKS

By

Ning Liu

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Computer Science and Engineering

2003

ABSTRACT

DIAS: DISTRIBUTED INTEREST-BASED ADAPTIVE SEARCH FOR P2P NETWORKS

By

Ning Liu

Content search in decentralized P2P networks is a challenging problem. Gnutella, a widely used P2P system, employs the query flooding approach. Although this approach is simple and robust, its efficiency is fairly low.

In this thesis, we propose an efficient P2P information retrieval system DIAS that supports state-of-the-art content search in the semantic layer. DIAS avoids the inefficient random flooding used in Gnutella by using an interest-based search scheme. Peers in a DIAS system sharing the common interest set are call "buddies". Each DIAS peer creates and updates its buddy list by computing the degree of interest similarity between its neighboring peer and itself periodically. Different interest groups are generated and maintained according to the buddy-relationship between any two peers. DIAS peers will issue and forward queries mostly within the interest groups associated with them.

Our simulation result verifies that DIAS could achieve satisfactory search efficiency without producing too more traffic overhead, while retaining the simplicity and fairness of Gnutella. The implementation of DIAS on top of JXTA shows that it is a feasible system.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Lionel M. Ni, for his continuous support and help throughout the research project. It was definitely my great pleasure to conduct this interesting and challenging research under his guidance. His broad and profound knowledge and his effective instruction have given me a great help.

I am also very grateful to my co-advisor Dr. Li Xiao, my academic committee members, Dr. Abdol H. Esfahanian and Dr. Matt W. Mutka, for their valuable advice and inspiring comments.

Many of my colleagues have contributions to my dissertation. I would like to thank Zhenyun Zhuang, Feng Zhu, and Pei Zheng for their insightful comments and suggestions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 Motivation

As the rapid growth of the Internet, P2P networks are gaining significant popularity in a quite short period of time. Centralized indexing, query flooding, and hybrid are the most widely used techniques in the present P2P systems to conduct content searching from any given peer on the overlay. Centralized indexing is employed in Napster [1] system, where peers automatically upload lists of available files to the central index, and queries are answered using this index. Flooding-based approach is applied in Gnutella [1] system, where queries are broadcasted over an overlay network connecting peers. Hybrid architecture is employed in Kazaa [1] system, where some peers are elected as "super-nodes" in order to index content available at peers in a nearby neighborhood and manage queries.

These approaches cannot provide deterministic guarantees because they do not make use of peer's knowledge. A knowledge-ware [3] system is a system in which information/knowledge have been represented uniformly and well organized into the knowledge base, and the resources can be wide-spread shared and coordinated used to provide on-demand and intelligent service satisfying personal needs and expectations. As we know that in searching the largely distributed data, the efficiency of a P2P Internet application mostly depends on the scalability and versatility of its search mechanism [4]. The current approaches are inefficient not only because they will generate a large amount of network

traffic for each query, but also because they do not utilize peer's knowledge to improve searching success rate.

In fact, there is an opportunity for many peers to identify a small set of neighbors that have a higher probability to answer the queries from the current peers. Based on this idea, many new systems have been developed to provide fast and reliable search and lower traffic overhead. PeerSearch [5] and NeuroGrid [6] are two of these systems that have high efficiency, scalability, and reliability. PeerSearch is a P2P information retrieval system that achieves both efficiency and determinism through an elegant combination of index placement and query routing [5]. NeuroGrid is an approach that decentralizes search to ongoing network activity. In a NeuroGrid system, each successive search changes the initiating peer's knowledge that is about the relationship information between contents and other peers in the network. And this peer will utilize the knowledge when it conducts a search next times [6].

In this thesis, we propose an interest-based [7], semantic-layered peers clustering scheme, DIAS, to improve searching efficiency without producing too more overall traffic. The design philosophy is motivated by the existing work of NeuroGrid, which retains the primary characteristics of Gnutella network: simple, robust, and decentralized. We find that in the P2P networks, if there are some particular pieces of content residing in a certain peer, then the queries issued by this peer are very likely to contain the keywords which already exist in these pieces of content. Furthermore, if peer A is interested in a special piece of content which is located in peer B, then it is quite possible that peer B will also have other pieces of content which peer A may be interested in. At the same time, it is very likely that peer B can also find the pieces of content which it is interested

in located in peer A. Those peers who share the same interest can gather to compose a group, and the queries issued by the peers in this group will be forwarded mostly within this group. So the searches will succeed in a shorter time. Based on this scheme, we propose our Adaptive Interest-based File Sharing system: DIAS.

1.2 Problem Statement

As we have mentioned in the last section, many P2P protocols such as NeuroGrid use knowledge base to improve search efficiency. We know that capturing the characteristics of web users is an important task for the web site designers. By mining web users' historical access patterns, some demographics and behavioral characteristics of web users could be determined. The navigation path of the web users, if available to the server, carries valuable information [8]. Also in P2P networks, if we deploy peers' historical search behaviors, mine their interest characteristics from the information stored in their local and contained in the messages issued or forwarded by them, we can establish some special relationships among the peers. Using these relationships, we can navigate the transfer of messages to optimize the search path, so that improve the search efficiency.

1.3 Definition of Terms

1.3.1 P2P Networks

In the traditional client/server architectures, some computers called servers are dedicated to serving the other computers, and some computers call clients are dedicated to being served by the server computers or server software. In a P2P network, each peer has equivalent capabilities and responsibilities. A P2P network is a type of network in which

two or more peers are connected and share resources, such as files and devices like scanners and printers without going through a separate server computer or server software.

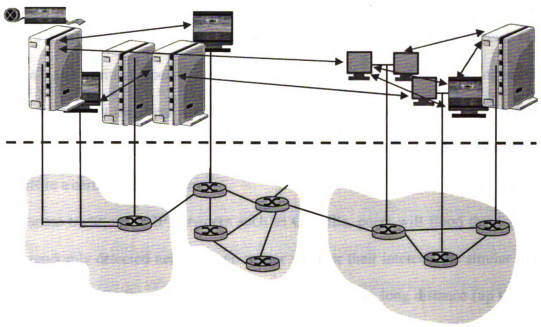

Figure 1: A logical architecture of a P2P network

Figure 1 shows the architecture of a typical P2P network. A P2P system is an overlay built on top of the physical infrastructure of a network. It can be a network of very small scale. For example, it can be a permanent infrastructure that consists of tens of computers and devices in an office or a small company. Or it can be much larger, such as a network in which different protocols and applications set up direct or indirect relationships among computers and devices over the Internet. Also a P2P network can be in an ad hoc environment — where network computers and devices are linked by wireless or temporary plug-in connections. In this case, the computers and devices are components of the network only for a certain period of time, for example, the duration of a communication session.

1.3.2 Interest Base

Among the users who are using the P2P network, it is very likely that many of them may be interested in some particular category of content. For example, some users may be very interested in "car" or "automobile", some users may have special taste in "movie", and "Britney Spears" may be the favorite singer star of many young users. It is quite possible that the users who have a special interest keep many pieces of content associated with this interest in their local computer, and they are very likely to search similar stuff in a certain period of time.

In some traditional P2P networks such as Gnutella, peers will flood their queries to some randomly detected neighbors no matter whether their interests are similar to or far from each other. So the query may be propagated to a very long distance (up to the TTL of a query), and the search efficiency cannot be satisfactory. The same case also occurs in another famous approach, random walk [9], where a query message is forwarded to a randomly chosen neighbor at each step until the object is found. We intent to design a scheme, through which the users who have the same or similar interest in a P2P network can be clustered into a group and connected to each other in a certain period of time. Then according to our common sense, if a user issues a query which is associated with his special interest and the query is forwarded mostly within the interest group of other users who also have this interest, then the wanted file or item will be found in a much shorter time compared with those traditional P2P networks. This is the basic idea of our DIAS system.

1.4 DIAS

An interest group is a set of nodes that show a high probability in querying the same category of content in a P2P system. For example, nodes that frequently send queries of Linux documents could be put into a "Linux Document" group. The protocol we propose intends to provide a mechanism for each node to determine which groups to join, how to join and leave, and its impact on content searching and retrieving.

The basic idea of DIAS is for any peer to select a number of current neighboring peers whose "interest index" is close to the current peer. We call these special neighbors of a peer "buddies" of that peer. The peers who share the same interests form interest groups on the overlay. The number of those neighboring peers selected is called "group size". Peers in this system will calculate their buddy sets periodically, using active polling scheme or passive reporting scheme, which will be detailed later. In addition to determining whether a neighboring peer is a buddy, a peer can also determine whether that neighboring peer's neighbors are its buddies, based on these peers' interest information. Then gradually buddies of same interest will be clustered in a group. Using this approach, after a period of warm-up time, a peer will be surrounded by its buddies on the overlay. And its query will be sent to these buddies directly. Also the queries coming into this peer will be forwarded to these buddies.

Each node in a P2P system is associated with a number of interest groups. In the case that multiple users may share the same node each user has a profile that defines a set of interest groups. While bootstrapping, a node is able to identify its interest groups based on its query history. Thus an initial *i-group* list could be created in the node. To partici-

pate in an unstructured P2P network, the node has to discover a small number of buddies who have the similar interest such that it can route query messages via these neighbors. A buddy list can be obtained from buddy cache from previous operation history, or gradually can be populated after some warming-up time. Each *i-group* entry of a node may have multiple ranked buddies. As a result, query search will have a large probability to success in the interest group.

1.5 Description of the Remaining Chapters

We will elaborate our DIAS system in the following chapters: In Chapter 2, we introduce some related works including Freenet, Gnutella, PeerSearch, Neurogrid, and JXTA. We present the architecture of our DIAS system in detail and show the simulation result in Chapter 3. In Chapter 4, we introduce the implementation of DIAS on JXTA platform. At last, we draw a conclusion on this thesis and discuss the future work in Chapter 5.

CHAPTER 2 RELATED WORK

2.1 Overview of P2P Protocols

2.1.1 Gnutella

Gnutella [2] is a fully decentralized, P2P application layer network designed by Nullsoft, a subsidiary of American Online. In a Gnutella network, each peer stores some special files and routes file searches (queries) from and to neighboring peers. If the searched file is stored locally in a certain peer, then this peer will response to this query, and the peer who initializes the query will download the file directly from the peer who has the file. The Gnutella network work by simple blind flooding: a peer who wants to search a file sends a query to its neighbors on the overlay. When the query is forwarded to a peer, and if this peer does not have the searched file, the query will be further forwarded to its neighbors, and so on, until a certain query TTL is reached. Each querying message has a global unique ID (GUID), and each peer maintains a list of recently seen GUID. Using this strategy, peers in Gnutella networks can filter out the messages they have already seen recently, so no messages can be received by a peer more than twice and query looping is avoided. In order to download the file that is found at some peer, each peer keeps a table locally that stores the information about where each searching query comes from. Using these tables, searching responses can be routed back to where they come from. At last, a connection will be established between the peer that issues the query and the peer who offers a response for the query, and the original peer can download the file from the

destination peer directly through HTTP. Figure 2 [18] shows how Gnutella works in each step.

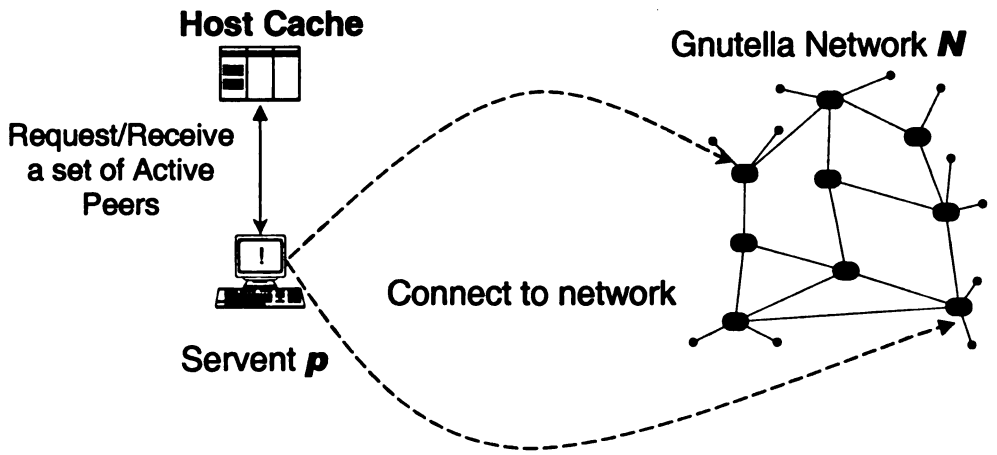

Figure 2: A new servent joining a P2P network

A Gnutella servent connects itself to the network by establishing a connection with another servent currently on the network. Once the address of another servent on the network is obtained, a TCP/IP connection to the servent is created. This step is shown in Figure 2.

Once a servent has connected successfully to the network, it communicates with other servents by sending and receiving Gnutella protocol descriptors.

A servent sends a Ping descriptor to actively discover hosts in the network. A servent receiving a Ping descriptor is expected to respond with one or more Pong descriptors, which includes the address of a connected Gnutella servent and information regarding the amount of data it is making available to the network. This step is shown in Figure 3.

A peer sends a Query descriptor to search a particular document in the network. A servent receiving a Query descriptor will respond with a QueryHit if a match is found

against its local data set. The QueryHit descriptor provides the recipient with enough information to acquire the data matching the corresponding Query. This step is shown in Figure 4.

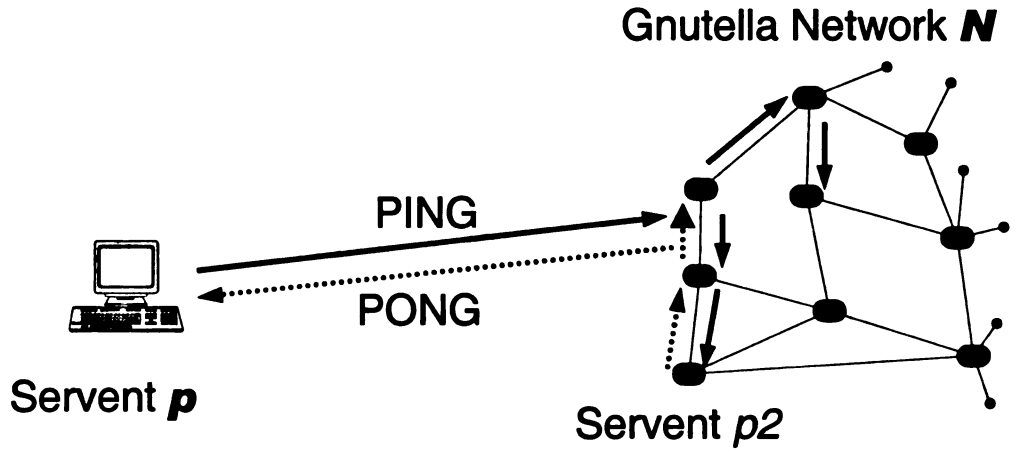

Figure 3: A servent discovers hosts in the network

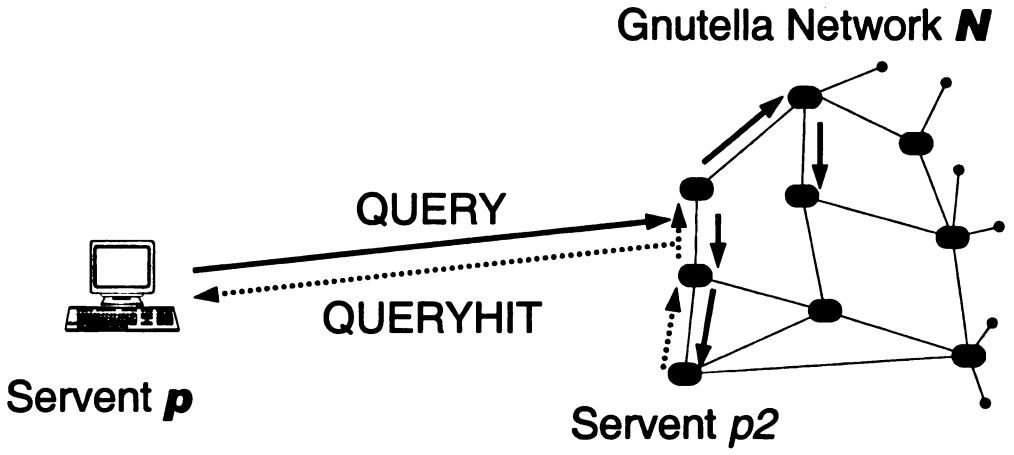

Figure 4: A servent requests a file

Gnutella works in a parallel way, in which the query will spread through the whole network as far as possible until the connectivity and TTL (Time-to-live) exhaust. If a

match or a possible loop is discovered in a peer, then the query will stop at this peer. But the search will still continue to propagate to the other peers in the network. Thus a large amount of traffic overhead will be generated even if a query response is available from a peer several hops away from the sender.

2.1.2 Napster

Napster [1] is a centralized application-level, client-server protocol over point-to-point TCP for sharing files over Internet. The Napster system works in the following steps: When a peer enters the system, it first connects to the Napster server, and then uploads its list of files to the server. This process is called "push". The peer will also send the server a set of keywords that can be used to identify its local files. When another peer sends query to the Napster server, the server pings the hosts that apparently have the data, and then select the best of the correct answers. Then this user retrieves the files it requires.

2.1.3 Freenet

Freenet [2] is another well-known system in P2P environments. In a Freenet network, each file is associated with a unique key that is created by hash functions and can be used to identify the file. At the same time, each peer has a knowledge base (KB) that stores information about the relationships between keys and peers. When a peer initializes a query, Freenet ranks the different keys it knows in the network, by comparing these keys with the key included in the query message, the more similar keys are ranked higher. Then the query will be forwarded to the peer that is associated with the most similar key. If the requested file cannot be found in this peer, then the query will be forwarded to the peer that is associated with the second most similar key, and so on. As we can see that

Freenet works in a greedy serial fashion. When a match occurs, that is, the search succeeds, the query is returned along the query path, and the searched file is passed to the original peer together with the query along the query path. At the same time, the found file is cached or replicated in all the peers along the querying route. Also to guarantee anonymity, any peers along the querying route can declare that they are the source of the found file randomly. The original peer that issues the query will update its knowledge base (KB) so that the key that it searched can be associated with the peers that declared to be the source of the found file. This will be quite helpful in the next time when the peer issue a query including the similar keys.

2.1.4 Kazaa

Kazaa [9] uses a hybrid architecture in which some peers are elected as "supernodes". Content available at regular peers surrounding a supernode are indexed.

To search a particular piece of content, regular peers send query requests to their supernodes. Supernodes communicate amongst themselves to find where the wanted data is located. When succeed, they return locations of matching objects to the requesting peer. To download an object, a peer initiates one or more connections to other peers that have replicas of the object. The requesting peers may either transfer the entire object in one connection from a single peer or choose to download multiple fragments in parallel from multiple peers. The download fashion in Kazaa is different with that in Gnutella, where only one connection is used. Thus Kazaa carries a more efficient download scheme.

2.1.5 NeuroGrid

NeuroGrid [6] is an adaptive decentralized network system. Peers in NeuroGrid system support distributed searches through semantic routing, that is, queries are forwarded based on content. A learning mechanism is applied in NeuroGrid system. It adjusts metadata describing the peers' content and the documents that consist of those contents. Like Freenet, each NeuroGrid peer also maintains a knowledge base (KB) locally. Any time when a search succeeds, the original peer that issues the query will update its knowledge base to store the information about the relationship between the document keywords and remote peer in which the match occurs. At the same time, a direct connection is established between the original peer and the destination peer. Each NeuroGrid peer forwards queries only to a subset of peers in which it believes the wanted file may be found, instead of forwarding queries to all of its neighbors. And this subset is determined by resorting the knowledge base. [2].

NeuroGrid system improves the searching efficiency and reduces the traffic overhead. Compared with Gnutella, NeuroGrid supports a really efficient search procedure because it is able to consistently determine peer identities. Compared with Freenet, NeuroGrid produces much fewer local cache overheads that are required to achieve some of its attractive qualities. But there is an important side effect in NeuroGrid system: Whenever a match occurs in a remote peer, a new connection will be established between the original peer and the remote peer, and a new record storing the relationship between the querying keywords and the destination peer will be added to the knowledge base of the original peer. So after a long time, too many new connections will be added to the network, and too many new records will be stored in each peer's knowledge base, but no old connec-

tions or old records will be deleted. This will ultimately be very expensive, with connections and knowledge bases growing indefinitely.

2.1.6 Interest-Based Locality Using Shortcut

This content location solution using interest-based locality principle [7] is proposed by a research lab in Carnegie Mellon University. In this solution, peers loosely organize themselves into an interest-based structure on top of the existing Gnutella network. The interest-based locality principle posits that if a peer has a particular piece of content that one is interested in, it is very likely that it will have other items that one is interested in as well. Shortcut discovery is used in this solution: When a peer joins the system, it first attempts to locate the content through flooding. The lookup returns a set of peers that store the content. These peers are potential candidates to be added to a "shortcut list". Subsequent queries for content go through the shortcut list and select the peers with the highest rank. If a peer cannot find content through the list, it issues a lookup through Gnutella, and repeats the process for adding new shortcuts.

2.1.7 PeerSearch

PeerSearch [5] is a P2P information retrieval system proposed by HP Lab. The basic idea of this system is to combine index placement and query routing. It only needs to search a small number of servants to identify matching documents. Leveraging some information retrieval algorithms, PeerSearch represents documents and queries as vectors and measure the similarity between a query and a document as the cosine of the angle between their vector representations. PeerSearch stores a document index using its vector

representation as the coordinates, resulting in that indices stored close to each other are also close in semantics.

2.2 Research Issues in Existing P2P Applications

By comparing these protocols, we can say that: Napster uses the centralized client-server model, which is not as flexible as the other protocols, and the additional server is a big consuming. Kazaa employs hybrid architecture where a few supernodes act as the servers. Gnutella is running a fairly inefficient search procedure in a fully decentralized network. Freenet, NeuroGrid, the shortcut solution, and PeerSearch appear to support a more efficient search process than Gnutella. But Freenet achieves this with much more local cache requirements, because it works in a greedy serial fashion. In NeuroGrid, new connections continue to be added among the peers in the network. New records continue to be inserted into the knowledge database of each peer. So the connection overhead and knowledge base overhead are really critical. The shortcut solution generates and maintains shortcut list for each peer according to the results of the searches issued by the peer, Peers in the list are believed to share the same interests with this peer. PeerSearch computes the similarity between document vectors and query vectors to optimize search process.

By studying the P2P protocols described above, we get the purpose of this thesis: To find a scheme that can improve the search efficiency in P2P networks while producing additional overhead as few as possible. The download efficiency also needs to be improved, but we won't cover this issue in this thesis. Our DIAS protocol can improve search efficiency without needing to concern the connection overhead and local informa-

tion overhead while producing no more traffic overhead. We achieve the enhancement by deploying the interest similarity between peers and generating interest groups.

2.3 Information Retrieval

The interest base algorithm of our DIAS system is built from the information retrieval conception. The problem of information storage and retrieval can be stated as: [10] we have vast amounts of information to which accurate and speedy access is becoming ever more difficult. The difficulty is not only how to extract the information but also how to use it to decide relevance.

The purpose of an automatic retrieval strategy is to retrieve all the relevant documents at the same time retrieving as few of the non-relevant as possible. When the characterization of a document is worked out, it should be such that when the document it represents is relevant to a query, it will enable the document to be retrieved in response to that query. When the indexing is done automatically it is assumed that by pushing the text of a document or query through the same automatic analysis, the output will be a representation of the content, and if the document is relevant to the query, a computational procedure will show this.

We will elaborate the conception of information retrieval in Chapter 3.

2.4 JXTA

As the collection of all software and devices connected to the Internet, the web is evolving from a Web of Computers to a Web of Things [11] as new devices are connecting to the Internet. Wireless devices, a new generation of web devices, such as pagers,

PDAs, cell phones, and other consumer devices are among these new devices that are entering the web world. Analysts predict that there will be a grand demand for wireless applications in the near future. P2P computing is leading itself to the more dynamic environments of wireless devices rapidly [12]. Because the P2P applications have the properties that they are capable of creating, joining, interacting with peer groups, and posting advertisements to provide and request resources, they can dynamically find the stuff that they need. This characteristic makes P2P applications suite to wireless environments quite well.

JXTA [11] is deployed based on the imagination of a world in which peers can take advantages of being connected with each other, despite what software and hardware platforms they are using. It is an open-source project that specifies a standard set of network protocol upon the existing physical network infrastructure for ad hoc, pervasive, P2P computing as a foundation of the upcoming Web of Things. A wide variety of P2P network applications can be built above this thin but powerful layer. The JXTA project aims at being independent of programming languages, system platforms, service definitions, and network protocols. It provides a uniform, addressable network based on assigning each peer in the network a unique peer ID, with all of the complexity of the underlying physical network topologies being hidden. So peers in the JXTA network are allowed to discover and communicate with each other, self-organize into groups, advertise and look for network resources, without being required to understand and manage the physical network topologies. Despite how complicated and dynamically the topologies are changing, JXTA peers do not need to care about.

CHAPTER 3 DIAS ARCHITECTURE

3.1 The Design of DIAS

The main idea of our DIAS system is to cluster peers on the overlay that share common interest set on the semantic layer based on the analysis of their local content, searching entities, and forwarding entities. Peers in such an interest group (internet zone) are called group members. The members in a group are connected together according to some scheme after a period of warm-up. And the quires issued by the group members in a group are to be forwarded mostly within this group. To determine whether two peers share the similar interest set, some vector space [14] based similarity coefficients have been investigated. Our preliminary simulation results show that such a semantic layer improvement to content searching for P2P networks will significantly improve searching efficiency with moderate overhead. Figure 5 shows an overview or our DIAS system.

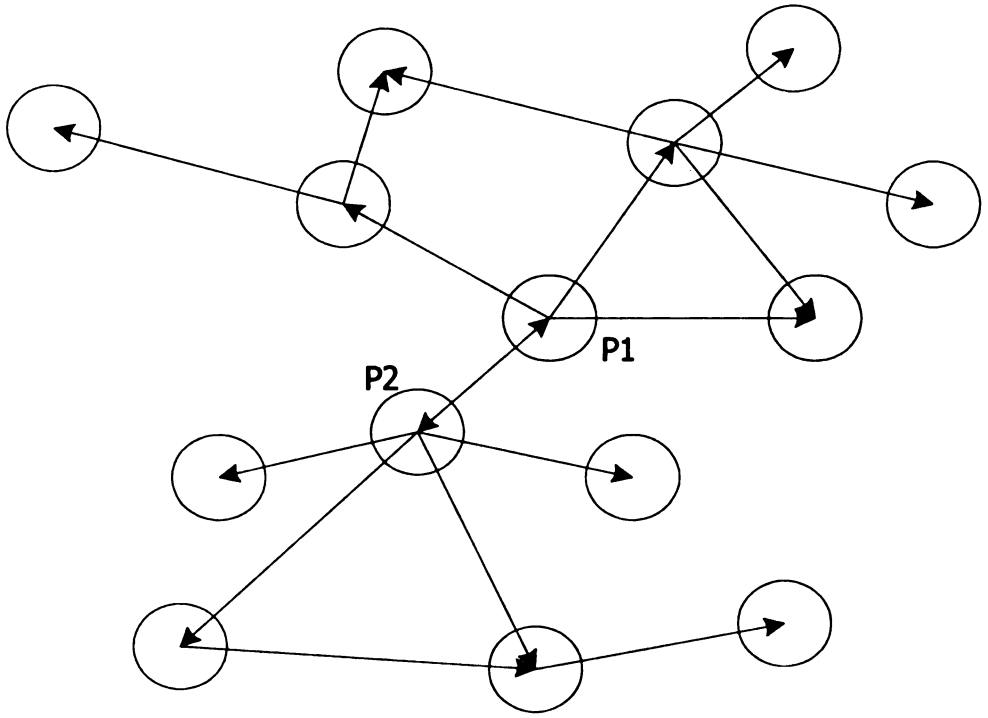

Figure 5: An overview of DIAS

3.1.1 Interest Database and Interest Index

Our DIAS is based on the analysis of the interest set of each peer in the P2P networks. Here we analyze the interest set according to the keywords that are contained in the files or documents existing in the P2P networks and the querying messages going around. In DIAS system, each peer maintains a local interest database that consists of three parts: LCK (Local Content Keyword), LQK (Local Query Keyword), and FQK (Forwarding Query Keyword). By comparing the keywords in these sets of each peer, our DIAS system can determine whether any two peers in a network share the common interest set, and according to the result of the comparison, each interest group can be formed.

Now let's take a look at these three keyword sets:

- LCK (Local Content Keyword): Keyword set of the local content of each peer. If a user has some special interest, then it is very likely to have some files or documents related to its interest stored in its local computer or device, that is its local content. So the keywords contained in the local content can indicate the interest of the user, or, say, the peer. In our simulation that will be described later, the local content of each peer will be assigned when the simulator starts, and LCK set will be created at the same time. In addition, when the peer receives a query reply from other peers, or retrieve some content from other peers, it will update its LCK set by putting the keywords of the content in the response that are not included in its LCK set into its LCK set. The maximum size of the LCK set is a parameter that may affect the performance of our protocol. When the size of the LCK set of a certain peer exceeds the specified maximum value, this peer will delete the oldest keywords in its LCK set and maintain the size less than or equal to the specified maximum value.

- LQK (Local Query Keyword): Keyword set of last M_1 number of queries. If a user has some particular interest, then it is quite possible that this user will issue the queries that are related to its interest frequently. So the keywords contained in the querying messages issued by a peer can also indicate the interest of this peer. In our simulation, anytime when a peer issues a query, it will update its LQK set. If some of the keywords contained in the querying message are not included in the LQK set of this peer, it will put those keywords into the its LQK set. The maximum size of the LQK set is also a parameter that may affect the performance of

our protocol. Each peer will maintain the size of its LQK set according to the same scheme used while maintaining its LCK set.

- FQK (Forwarding Query Keyword): Keyword set of last M_2 number of forwarded queries. According to our analysis to the behaviors of peers in P2P networks we believe that some times, the keywords contained in the queries that a certain peer often forwards may also indicate the interest of this peer. So in our simulation, we considerate this factor in a few cases. Anytime when a peer forwards a query, it will update its FQK set according to the same scheme described above.

Peers in our DIAS network system use these three keyword sets to estimate the interest of each, look for their buddies, and generate interest groups. In the next section, we will talk about how DIAS peers generate and maintain their buddy lists using these three keyword sets.

3.1.2 Buddy List

A peer's buddy list contains neighboring peers in the interest group to which the peer should send its query. A peer decides whether one of its neighboring peers is its buddy by comparing the LCK, LQK, FQK keyword sets of that peer and itself, so the peers in its buddy list shares the common interest set with it. The average search path length in our DIAS network system can be reduced considerably with each peer sending and forwarding queries directly to its buddies.

The buddy list of each DIAS peer has to be updated from time to time in order to reflect the most recent interest distribution in the network. We provide two update schemes: polling and reporting. The simulation can run based on either of these schemes.

- **Polling**

A peer will update its buddy list every BUDDY_UPDATE_LOOP times of its issuing queries. Here BUDDY_UPDATE_LOOP is a parameter in our properties file that will affect the performance of the system. For example, if BUDDY_UPDATE_LOOP is set to 10, then a peer will update its buddy list when it issues 10 queries, 20 queries, 30 queries, etc.

When a peer is about to update its buddy list in the interest group, it will send interest-PING message with specific indication of LCK, LQK, or FQK to its neighboring peers. Its neighbors will reply to the interest-PING with their local interest databases. The information of the interest database of other peers will be saved (cached) in the underlying peer for later use.

To go one step further (more step could be possible, which is determined by TTL of interest-PING), when a peer is about to update its buddy list, it can also request interest information of the peers who are the neighbors of its direct neighbors. For such request, a peer can either reply with its local cached interest information, or initiate another interest-PING with TTL set to 1 to its own neighbors, which in turn updates its local cached interest information. To avoid flooding when establishing buddy relationship for a peer, the interest-PING should have a TTL of less than 3. Again this parameter will likely affect the performance of the system.

We cannot directly exchange buddy list between two peers due to the fact that they might be in different interest groups. Instead, upon receiving the interest information of its neighbors and its neighbors' neighbors, a peer can decide which one among these peers can be added to its buddy list following some "Interest Matching" algorithms that

will be described later. When a peer identifies a buddy after calculation, and this buddy is not connected to this peer right now (the buddy must be the neighbor's neighbor of this peer), then a new connection will be established between the buddy and the polling peer. As a result, the connections of each peer will become more and more gradually, and this is an overhead to our system. The maximum number of the connections of each DIAS peer is a parameter that may affect the performance of our protocol. When the number of a certain peer's connections exceeds the specified maximum value, this peer will first delete the connections of its neighbors that are not its buddies, and then, if its connections are still more than the specified maximum value, it will delete the buddies with the lowest rank in its buddy list and maintain its connections less than or equal to the specified maximum value. We will describe how the buddies are ranked for a certain peer in Section 1.3.

- **Reporting**

In the reporting pattern, DIAS peers will not inquire the interest database information from their neighbors positively as in the polling pattern. Instead, they wait passively. Any time after a peer updates its buddy list, it will check the difference between its old buddy list and its new buddy list, that is, the number of the peers that are in the old buddy list but not in the new buddy list plus the number of the peers that are in the new buddy list but not in the old buddy list. If the total number exceeds a specified maximum number NO_CONN_CHANGES then this peer will report to its neighbors. Here NO_CONN_CHANGES is a parameter in our property file that may affect the performance of our protocol.

A peer reports its buddy list change in such a way: It sends a message to all of its neighbors telling them that its buddy list has been changed enough for them to considerate, and the neighbors may forward this message to their neighbors, according to the TTL of interest-PING. Again, as in the polling fashion, to avoid flooding, the interest-PING should have a TTL value less than 3. And this parameter will likely affect the performance of the system.

Each DIAS peer maintains a variable "o_no_neibor_changes" that records how many messages described above have been sent to this peer. If the number of the received messages exceeds a specified maximum number NO_BUD_CHANGES then this peer will update its buddy list. Here NO_BUD_CHANGES is a parameter in our property file that may affect the performance of our system. After the peer updates its buddy list, it will report and set the variable "o_no_neibor_changes" to 0 to start a new turn of recording.

3.1.3 Interest Matching

- **Information Retrieval**

Definition

In order to elaborate the scheme that we used in our DIAS network system for the interest similarity measure, let's first take a look at the concept of information retrieval based on which we developed our interest matching strategy.

What is information retrieval (IR)? It is a wide loosely-defined term. Lancaster [15] gave a perfect and straightforward definition: "Information retrieval is the term conventionally, though somewhat inaccurately, applied to the type of activity discussed in this volume. An information retrieval system does not inform (i.e. change the knowledge of) the user on the subject of his inquiry. It merely informs on the existence (or non-

existence) and whereabouts of documents relating to his request." Some basic concepts in the information retrieval term are as follows:

- Document is represented by a set of keywords. The keywords can also be call "index terms".
- Keywords may be assigned binary weights or gain their weighs from the calculation on statistics of their frequency in documents.
- "Retrieval" is a "matching" process between the keywords in documents and the keywords in queries.

The purpose of a retrieval strategy is to retrieve the relevant documents as many as possible and at the same time, retrieve the non-relevant as few as possible. After the characterizations of a document is mined out, when a document represented by these characterizations is related to a query, this document should be able to be retrieved in reply to that query. Below is a figure describing a typical information retrieval system [10]:

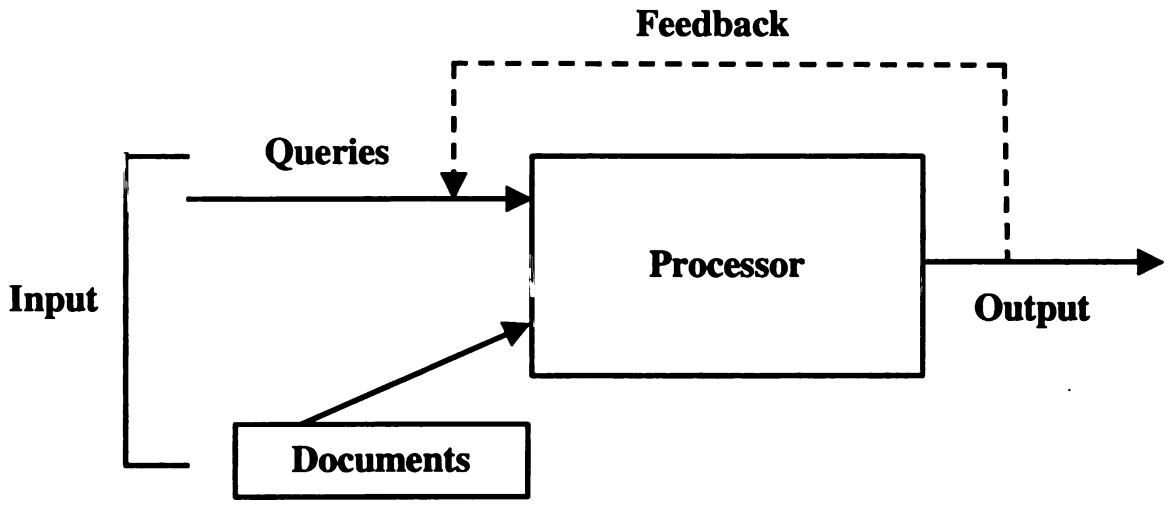

Figure 6: A typical information retrieval model

Model

Now let's embody the theory of information retrieval using a model. A typical IR model is shown in Figure 6. The purpose of an information retrieval model is to formalize the way of thinking about the information retrieval. In the model, we define a set of objects and assertions can be made about these objects. Also in the model, we restrict the ways in which classes of objects can interact with each other. The purpose of an information retrieval model is to specify the representations that are used for documents and information needs, and how the comparison between the keywords of them is implemented.

An information retrieval model can be presented as a $[D, Q, F, R(q_i, d_j)]$ where

- D is a set of representations for the documents in the collection
- Q is a set of representations for queries, that is, the user information needs
- F is a framework for modeling document representations, user information needs, and the relationships between them
- $R(q_i, d_j)$ is a ranking function. It associates a real number with a query representation $q_i (q_i \varepsilon Q)$ and a document representation $d_j (d_j \varepsilon D)$ (Baeza-Yates & Ribeiro-Neto, 1999 [16])There are several information retrieval models that have been deployed: Boolean Model, Vector Space Model, Probabilistic Model, Language Model, etc. The interest matching scheme used in our DIAS system is based on the vector space model, which is an algebraic model. In the vector space model, documents

ments in the collection (D) and user information needs (Q) are both documents in L-dimensional space.

Vector space model

Vector space model is based on the idea of L-dimensional document space. User information needs in this model are also located in document space. Documents and queries are both points in L-dimensional space, where L is the number of unique index terms in the data collection, and documents and queries are vectors of these index terms. Actual vectors may have many terms, and can use binary keyword weights or assume 0-1 weights. Below is a simple example:

Terms: "duck", "horse", "dog", "squirrel", "pig", "cat"
Binary: (1, 0, 0, 0, 0, 1), (0, 1, 0, 1, 0, 0)
Weighted: (0.01, 0.03, 0.01, 0.0, 0.05, 0.0)

A two-dimension vector space and a three-dimension vector space are shown in Figure 7 and Figure 8.

In the vector space model, not only documents can be weighted, but queries can also be weighted. If we assume the index terms of documents have different values for retrieval, then we can assign weights to each term in each document.

The vector space model calculates degree of similarity between documents and queries in the collection, and ranks output by sorting the values of similarity. Documents are ranked according to how close they are to the query. There are many possible matching functions in the vector space model and we will talk about them later. Figure 9 illustrates a simple structure of the vector space model.

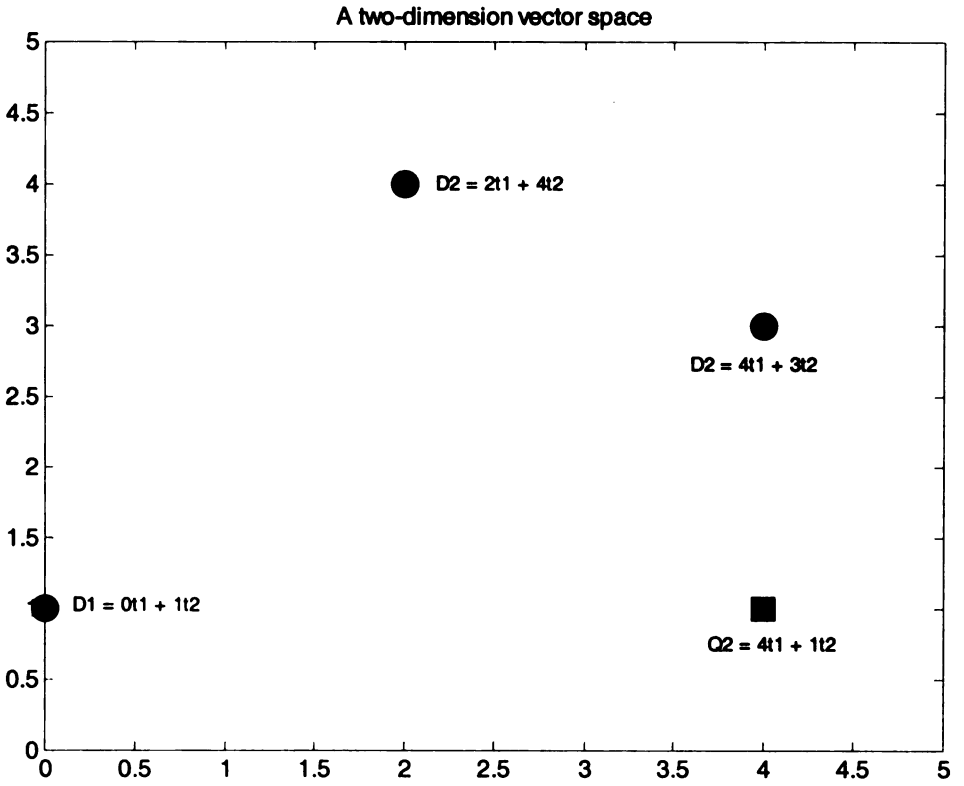

Figure 7: A two-dimension vector space

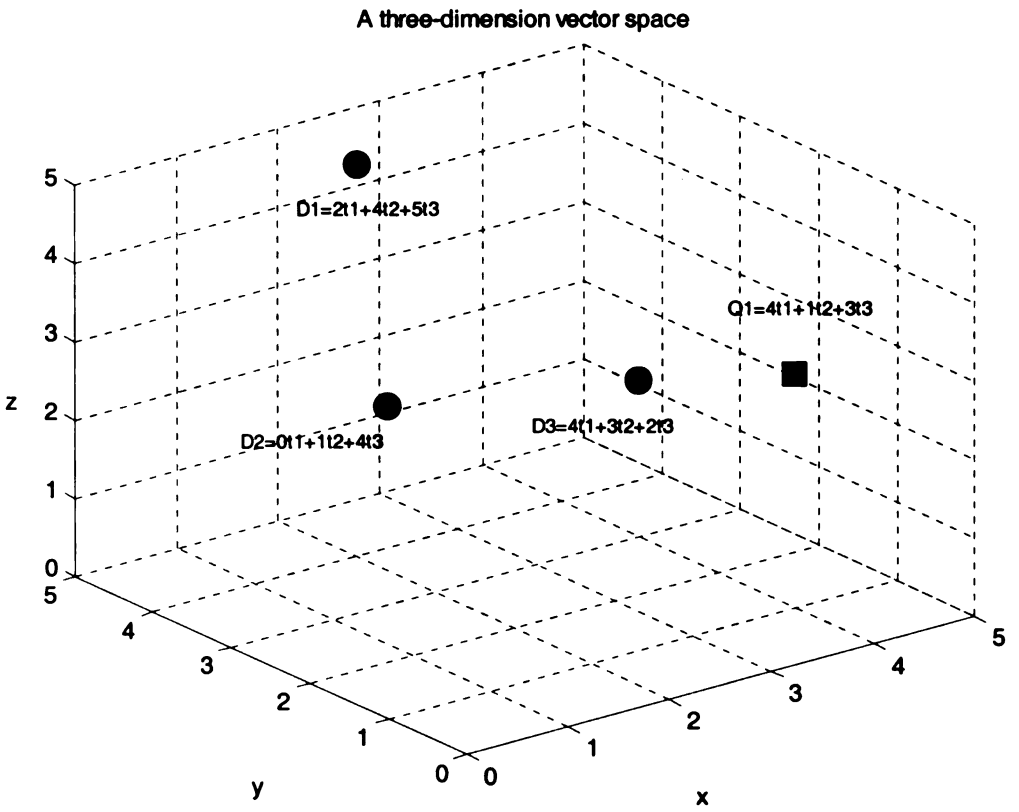

Figure 8: A three-dimension vector space

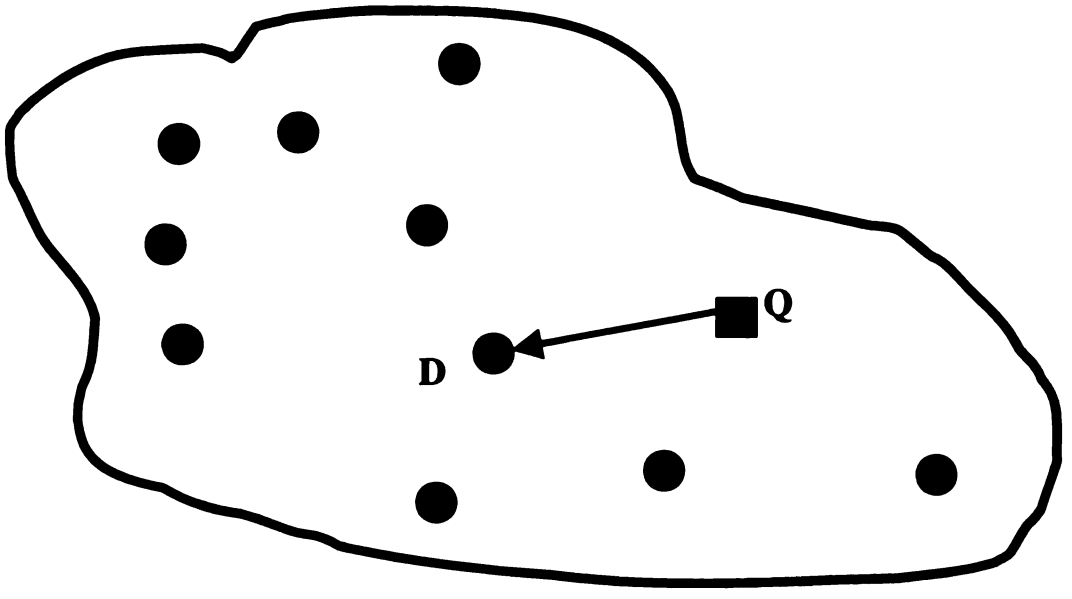

Figure 9: A simple structure of the vector space model

- **Distributional Similarity Measures**

In the vector space, the closer two vectors are, the more similar they are. Documents and queries clustering together are very likely to be highly similar. We need a method that allows ranking, using threshold and reformulating query, to calculate the similarity between documents and queries or documents and documents. There are several existing distributional similarity measures in the vector space model: Cosine coefficient [17], Jaccard coefficient [17], Dice coefficient [17], Overlap coefficient, L1 distance (City block distance, Manhattan distance), Euclidean distance (L2 distance), Hellinger distance, Information Radius (Jensen-Shannon divergence), Skew divergence Cosine coefficient, Jaccard coefficient and Dice coefficient are the most popular measures. We will only

elaborate these three measures that are based on contribution to similarity of co-occurring terms.

Some conventions are assumed first:

- N : the number of documents in collection
- L : the number of unique terms in collection
- D_i :the i th document in the collection
- T_i : the i th term in the collection
- Q : a query
- W_{i,D_j} : the weight of term i in Document j
- $W_{i,Q}$: the weight of term i in Query

Most similarity functions are based on the inner product. The similarity between a document and a query is the sum of the products of the term weights in the document and the query:

$$SIM(D_j, Q) = \sum (W_{i,D_j} * W_{i,Q})$$

And the similarity between two documents is the sum of the products of the term weights in these two documents:

$$SIM(D_i, D_j) = \sum (W_{k,D_i} * W_{k,D_j})$$

Dice coefficient

Using dice coefficient measure, the degree of similarity between a document and query is calculated using the equation below:

$$SIM(D_j, Q) = \frac{2\sum(W_{i,D_j} * W_{i,Q})}{\sum W_{i,D_j}{}^2 + \sum W_{i,Q}{}^2}$$

The sum is taken over all terms for index 1 to the dimensionality of the vector space L in the collection. For binary documents, the equation can be simplified to:

$$SIM(D, Q) = \frac{2A}{B + C}.$$

Jaccard coefficient

Using dice coefficient measure, the degree of similarity between a document and query is calculated using the following equation:

$$SIM(D_j, Q) = \frac{\sum(W_{i,D_j} * W_{i,Q})}{\sum W_{i,D_j}{}^2 + \sum W_{i,Q}{}^2 - \sum(W_{i,D_j} * W_{i,Q})}$$

The sum is also taken over all index terms for 1 to L in the collection. For binary documents, the equation can be simplified to:

$$SIM(D, Q) = \frac{A}{B + C - A}.$$

Cosine coefficient

Cosine coefficient shown in Figure 10 is the most commonly used measure. This measure is based on the calculation of cosine of the angle between a document vector and a query vector or two document vectors.

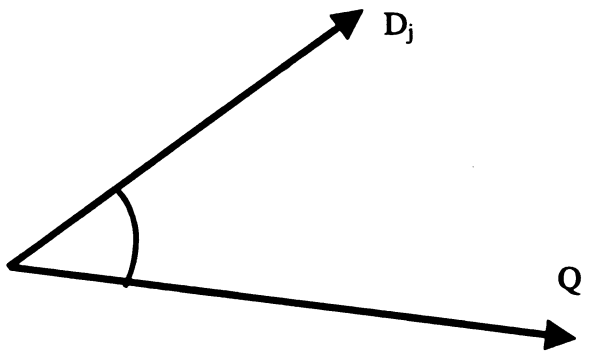

Figure 10: Cosine coefficient

Using this measure, the degree of similarity between a document and query is calculated using the equation below:

$$SIM(D_j, Q) = \frac{\sum (W_{i,D_j} * W_{i,Q})}{\sqrt{\sum W_{i,D_j}^{\ 2}} * \sqrt{\sum W_{i,Q}^{\ 2}}}$$

Again the sum is taken over all index terms for 1 to L in the collection.

- **Interest Matching in DIAS**

In our DIAS system, each peer has three keyword set, LCK, LQK, and FQK. In order to generate the buddy list for each peer, we need to calculate the degree of similarity between the keyword sets of two peers. Our computation is based on the cosine coefficient measure in the vector space, and some appropriate modifications are made according to the real case in our system. The Unions of these three sets are used in the computation. We know that in the standard cosine coefficient measure, the convention W_{i,D_j} denotes the weight of term *i* in Document *j*. In our DIAS system, we use the function $I(s_{nj}, k_i)$ to denote the existence of keyword *i* in the keyword set *j* of peer n. Here i

takes the value over all the keyword index in the keyword collection of our system, j takes the value from 1, 2, 3, because we only set up three keyword sets in our system, and n is the index of DIAS peer. If k_i exists in the joined keyword set j of peer n, then the value of $I(s_{nj}, k_i)$ is set to 1; otherwise 0. Let p_1 and p_2 be two peers in the DIAS system. Because the contribution to the total degree of interest similarity between two peers by each keyword set is different, we assign different weight to different set. The weight for each keyword set is denoted as W_j, where j = 1, 2, 3 for each set respectively. The degree of interest similarity between two peers can be computed using the following equation:

$$SIM(p_1, p_2) = \sum_{j=1,2,3} W_j * \left(\frac{\sum_i \left(I(s_{1j}, k_i) * I(s_{2j}, k_i) \right)}{\sqrt{\sum_i I(s_{1j}, k_i) * \sum_i I(s_{2j}, k_i)}} \right)$$

An example of Interest Matching computation is described in Table 1 and Figure 11. Here the keyword collection is {10, 11, 12, 13, ... , 22}, W_1=0.6, W_2=0.3, W_3=0.1.

Using the give equation, the degree of interest similarity between p_1 and p_2 is calculated as:

$$SIM(p_1, p_2) =$$

$$0.6 * \frac{2}{\sqrt{4*2}} + 0.3 * \frac{1}{\sqrt{2*2}} + 0.1 * \frac{2}{\sqrt{3*2}} = 0.654$$

Table 1 An example of two-peer interesting matching

Keyword Set	LCK	LQK	FQK
p_1	10,11,12,15	15, 17	20,21,22
p_2	10,11	15,20	21,22
$SIM(p_1, p_2)$	0.654		

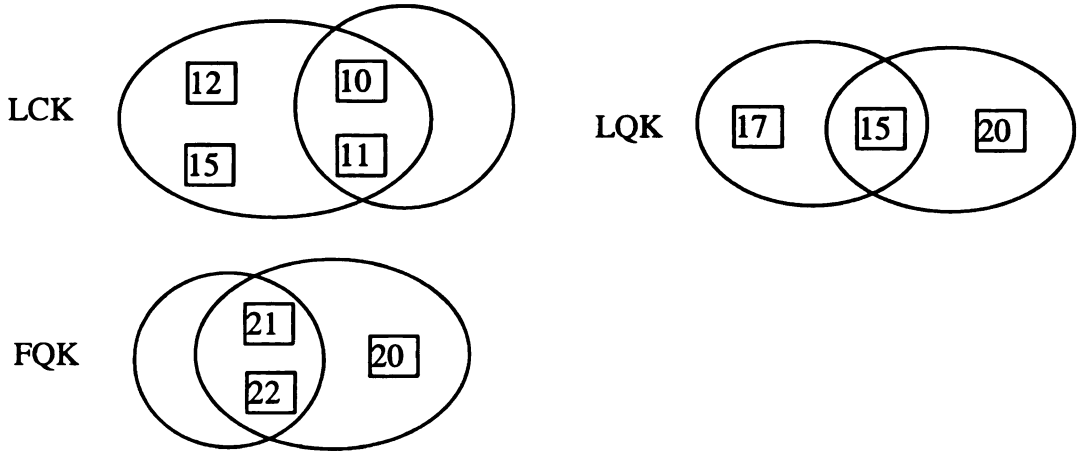

Figure 11: An example of two-peer interesting matching

After the calculation of the interest similarity between two peers, how can we determine whether these two peers are buddies of each other? We use a threshold to accomplish this. If the result of similarity calculation between two peers is equal to or more than the threshold, that means, there is enough number of shared keywords between them, then these two peers are considered as buddies, otherwise, not. This threshold is a parameter in our properties file and may affect the performance of our protocol.

Buddies in the buddy list of a certain peer are ranked according to the result of similarity calculation between the buddies and this peer. The buddy with a high value of result is ranked high, and the buddy with a low value of result is ranked lower, because the former

one has the more similar interest with the peer than the later one. When a peer has to delete some of its buddies because the number of its connections exceeds the limitation, it will first delete the buddies with the lowest rank that has the least similar interest with it.

3.1.4 Search Strategy

We have elaborated the core of our DIAS system: How the calculation of interest similarity between peers is implemented and how the buddy list of each peer is generated. We know that the main purpose of this thesis is to improve the search efficiency, and what we have done above are all for this purpose. Now let's take a look at how searches are implemented to improve the efficiency in the DIAS system.

Queries in DIAS network use similar techniques to Gnutella to prevent loops and indefinite propagation: We use TTL (Time To Live) counters to limit the range of queries, whenever the path length of a query exceeds TTL, this search will terminate despite whether it succeeds. We use GUID (Globally Unique Identifier) to prevent a node forwarding a query more than twice. Whenever a querying message arrive at a certain peer, this peer will first check whether it has seen the GUID of this querying message. If so, it will discard it; otherwise, it will process it.

In our simulation, each search is started at a randomly selected peer. When a peer is selected to initiate a query, a document will be selected randomly from the local content of this peer and a keyword will be selected randomly from the selected document. Then all the documents contained the selected keyword in the document collection of our DIAS system will be picked out except those in the local content of the selected peer. And the target document that the selected peer wants to search is selected randomly from the picked document set. Finally, the keyword set contained in the target document and

the target document itself are put into the corresponding fields of the querying message and the message is added into the inbox of the selected peer that will initiate a query.

After the querying message is generated, the selected peer will first update its keyword set LQK by adding the keywords in the query into LQK if they are not already in LQK. Then it will increase the number of searches it has initiated in the present turn. After these two steps, it will decide whether it's the time for it to update its buddy list based on the polling scheme or reporting scheme, and then update its buddy list using the method we described above if appropriate. Upon finished all of these steps, the peer will send its query to all of its neighboring peers, including its buddies.

Whenever a peer receives a querying message, it will first check whether the target document contained in the message reside in its local content. If so, the search succeeds. If not, the peer will update its keyword set FQK by adding the keywords contained in the message into FQK if they are not already in FQK. Then it forwards the message to all of its neighbors.

3.2 Implementation of DIAS

Our DIAS system is developed from the existing work of NeuroGrid. The system is built on the platform of Unix, using Java programming language. The functions are packaged into three major classes: *Node, Network, DishonestNode and DishonestNetwork.*

Major members in the class *Nodes* are:

- *protected String o_node_ID*: The unique ID of each node
- *protected Hashtable o_conn_list*: List of connections to other nodes
- *public MultiHashtable o_contents*: Stores docs indexed via keyword

- *protected Vector lck, lqk, fqk*: The three keyword sets
- *protected Vector o_buddy_set*: Buddy list
- *public int o_no_search*: The number of conducted searches in a certain interval
- *public int o_no_neibor*_changes: The number of neighbors who have updated their buddy list recently which is used in the reporting fashion.

Major functionality methods in the class *Node* are:

- public void UpdateKeywordSet(Vector p_vector, Keyword[] p_keywords): Update each keyword set according the parameter.
- public void add_buddy(Node p_node, int p_rate): Add a buddy to the buddy list
- public void IncreNoSearch(): Increase the value of the member *o_no_search* by 1.
- public void IncreNoChanges(): Increase the value of the member *o_no_neibor*_changes by 1

Major functionality methods in the class *DishonestNode* are:

- *public void UpdateBuddySet(int p_update_loop, double threshold)*: Update the buddy list
- *public void report():* A node report to its neighbors that it has updated its buddy list in the reporting fashion.
- *public boolean processMessage(Message p_message, boolean p_start) throws Exception*: Process a search message when it comes.

Major members in the class *Network* are:

- *public static double o_threshold:* The threshold used when generating the buddy list for each node.

- *public static double o_lck_coeff, o_lqk_coeff, o_fqk_coeff:* The coefficient used when generating the buddy list.

Major functionality methods in the class *DishonestNetwork* are:

- *public String[] searchNetwork(int p_start_TTL) throws Exception:* Initiate a search
- *public void generateContent(Hashtable p_nodes, int p_no_documents_per_node) throws Exception*: Assign documents to nodes using Zipf distribution

3.3 Simulation Setup

3.3.1 Parameters

In this thesis, preliminary simulations of the DIAS and Gnutella systems were performed. We set the parameters of these two systems in our properties file. Both of these two networks contained 1000 nodes that is set by the parameter NO_NODES in the properties file, a pool of 2000 documents that is set by the parameter NO_DOCUMENTS and a pool of 1000 keywords that is set by the parameter NO_KEYWORDS. Here the nodes we refer to are equal to the peers we refer to above. Each document in the document pool is randomly assigned 1 keyword from the keyword pool. The reason why we set the parameter "NO_KEYWORDS_PER_DOCUMENT" to the value of 1 is because that by doing so, the distribution of the documents among the nodes represented the distribution of the keywords among the nodes. So we implemented the Zipf distribution of the keywords by distributing the documents in Zipf fashion. In the next section, we will elaborate how the documents were distributed among the nodes in Zipf fashion. Each node was connected to 30 other nodes that were selected at random. This is set by the parameter

NO_CONNECTIONS_PER_NODE. In both simulations, each node could receive information about the contents of their neighbor nodes. The initial TTL was set to 9 by the parameter START_TTL. Both simulations were run for 100,000 searches set by the parameter INTERNAL_LOOP, with each search being started at a node selected at random. The target of each search was for a document that shared the only keyword with documents in the starting node, which has been described in Section 1.4. The simulations were run in both polling fashion and reporting fashion. The number of searches by which a node will update its buddy set in the polling fashion is set by the parameter BUDDY_UPDATE_LOOP. The number of connection changes by which a node will report its change to its neighbors in the reporting fashion is set by the parameter NO_CONN_CHANGES, and the number of neighbor changes by which a node will update its buddy set in the reporting fashion is set by NO_BUD_CHANGES.

The properties file is shown in Figure 12.

```
SIMULATION_TYPE=neurogrid

MAIN_LOG_FILE=ng_growing_simulation.log
SUMMARY_LOG_FILE=ng_growing_sum_simulation.log
ARCHITECTURE_LOG_FILE=ng_growing_arch_simulation.log
PROBE_LOG_FILE=ng_growing_probe_simulation.log
BUDDY_LOG_FILE=ng_growing_buddy_simulation.log
SEARCH_LOG_FILE=ng_growing_search_simulation.log
APPLET=false

NO_KEYWORDS=1000
NO_DOCUMENTS=2000
NO_KEYWORDS_PER_DOCUMENT=1
NO_NODES=1000
NO_HONEST_NODES=990
NO_DOCUMENTS_PER_NODE=3
MAX_KNOWLEDGE_PER_NODE=1
NO_CONNECTIONS_PER_NODE=30
MAX_CONNECTIONS_PER_NODE=30
START_TTL=9
DEGREE_OF_CORRELATION=1

FORWARDING_MODEL=0
INTERNAL_LOOP=100000
PROBE_LOOP=800000
NO_PROBES=0
GROWTH_LOOP=100001
STATS_LOOP=1000

KNOWLEDGE=false
LEARNING=false
RING_TOPOLOGY=false
DOC_KEYWORD_ZIPF_DISTRIBUTION=false
NODE_DOC_ZIPF_DISTRIBUTION=true
RANDOM_SEARCHES=false
RANDOM_FORWARDING=false

THRESHOLD=0.4
LCKCOEFF=0.6
LQKCOEFF=0.4
FQKCOEFF=0.0
BUDDY_UPDATE_LOOP=30
NO_CONN_CHANGES=5
NO_BUD_CHANGES=5
NO_GROUPS=50
```

Figure 12: Properties file

3.3.2 Zipf Distribution

The term Zipf is used to describe the phenomena in which large events are of low frequency, but small ones are of quite high frequency. For example, large earthquakes seldom occurred in the world but small ones often occurred. Only a few words, such as 'is', 'are' and 'the' are used very frequently, but many other ones such as 'phenomena', 'earthquake' and 'astronautics' are rarely used.

The definition of Zipf's law can be give as "The probability of occurrence of words or other items starts high and tapers off. Thus, a few occur very often while many others occur rarely". Zipf's law usually refers to the 'size' y of an occurrence of an event relative to its rank r [18]. We can use the following equation to denote the frequency of the occurrence of the n^{th} ranked item:

$$P_n \sim \frac{1}{n^a}$$

In our DIAS system, the distribution of documents among nodes follows Zipf fashion. We assigned documents to nodes in two steps: First, the nodes were divided into several groups, because in the real world, it is very possible that users with different interest cluster into different groups. The number of groups was set by the parameter NO_GROUPS that may affect the performance of our system. Second, within each group, documents were assigned to nodes following the Zipf distribution. A few documents occurred frequently, others seldom. The documents highly ranked were selected randomly, so in different groups, different sets of documents occurred frequently.

Because the distribution of documents represents the distribution of keywords among nodes, which has been mentioned in the last section, keywords are also distributed in the fashion described above.

3.4 Simulation Results

3.4.1 DIAS vs. Gnutella

In this simulation, we set the value of NO_GROUPS to 50 for both networks, the value of THRESHOLD to 0.4 and the value of weight LCKCOEFF, LQKCOEFF, FQKCOEFF to 0.6, 0.4 and 0.0 for our DIAS network. The value of BUDDY_UPDATE_LOOP is set to 30 for the polling fashion. The value of NO_CONN_CHANGES is set to 5 and the value of NO_BUD_CHANGES is set to 5 for the reporting fashion. The results of simulations are shown in Figure 13-19.

From the data in Figure 16 we can see that the average path length was reduced dramatically in our DIAS system compared with the traditional Gnutella system from around 2.2 in Gnutella to around 1.6 in the polling fashion and around 1.3 in the reporting fashion of DIAS. The decreasing rate is 27.3% and 40.9%. At the same time, Figure 17 shows that the additional overhead resulting from the generation and maintenance of the buddy lists was quite small. The number of message transfers was increased from $3*10^4$ in the Gnutella network to only $3.02*10^4$ in the polling fashion of our DIAS system with increasing rate 0.67%. And in the reporting fashion of our system, the average traffic is almost not increased, except in the warm-up time at the very beginning.

We can also see that the performance in reporting fashion is better than that in polling fashion from the data shown in Figure 16: Gnutella vs. DIAS: Average Path Length. Re-

porting fashion improved the searching efficiency more grandly than polling fashion. It's because in the reporting fashion, only the node with the number of neighbors that had updated their buddy lists big enough would update its buddy list, and once the number met the requirement, the node would update its buddy list, despite how many searches it had conducted. And we know that the buddies of a certain node are among its neighbors, two nodes that have similar interest can be connected by the introduction of their intermediate buddies. So the updating of buddy list was quite efficient in the reporting fashion, thus nodes sharing the common interest set could be clustered together in a higher level. This caused the high searching efficiency. But in the polling fashion, each node updated its buddy list in a stable interval, despite whether necessary, so the updating efficiency is lower. Also the reporting fashion produced less traffic than the polling fashion. That's because in the reporting fashion, interest groups were generated more quickly in the warm-up time, and after that, nodes could not update their buddy lists easily for the sake of the limitation to the number of their neighbors that had updated their buddy lists. So only in the warm-up time, the additional traffic was a little much. But in the polling fashion, nodes continued to update their buddy list with rhythm, so the additional traffic existed all the time.

From these simulation results we can conclude that our DIAS system can improve the search efficiency considerably with the acceptable additional overhead in P2P networks in contrast to Gnutella system.

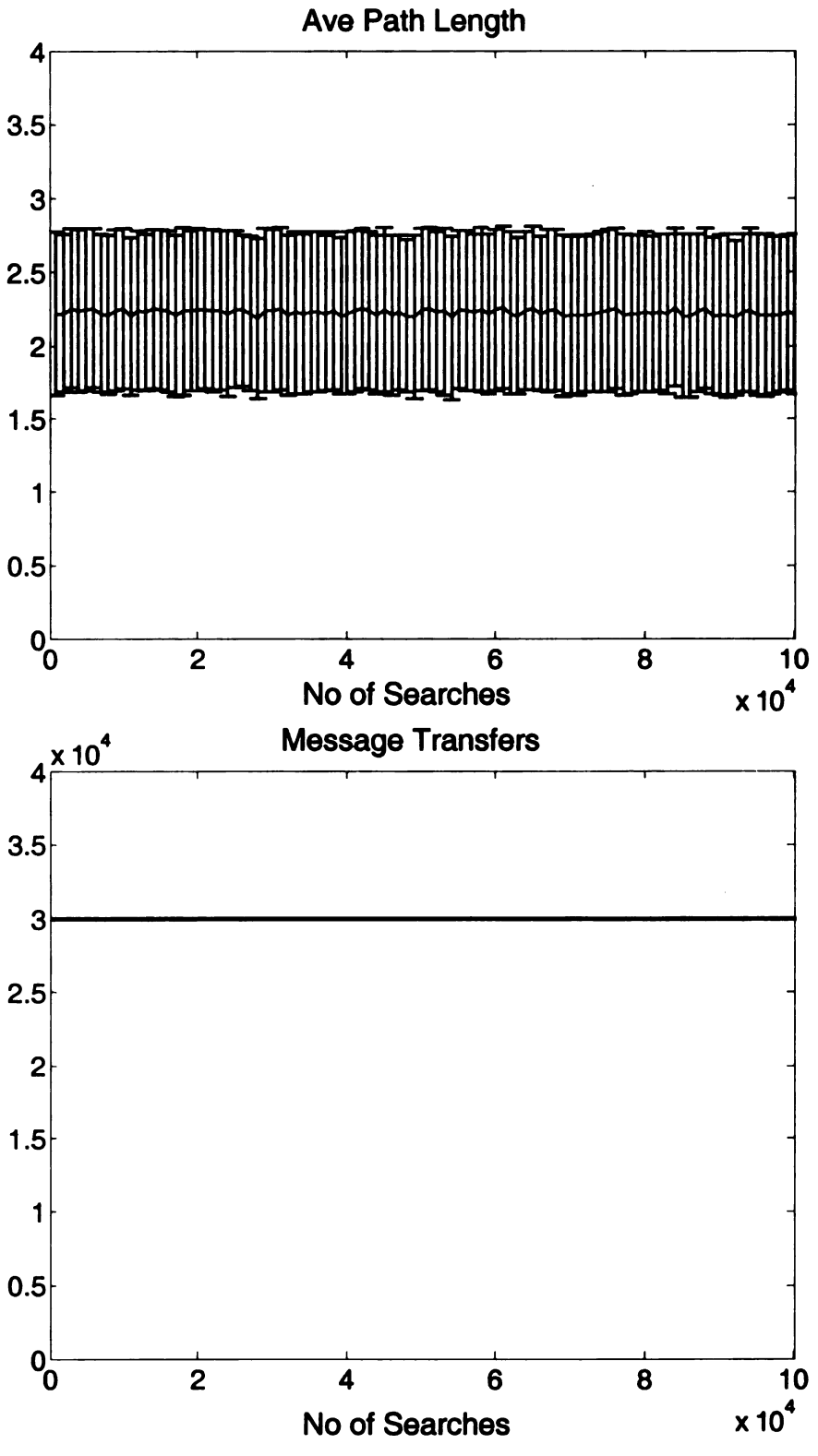

Figure 13: The 100000 searches in a 1000 node network, using Gnutella routing for a Zipf distribution of documents over nodes.

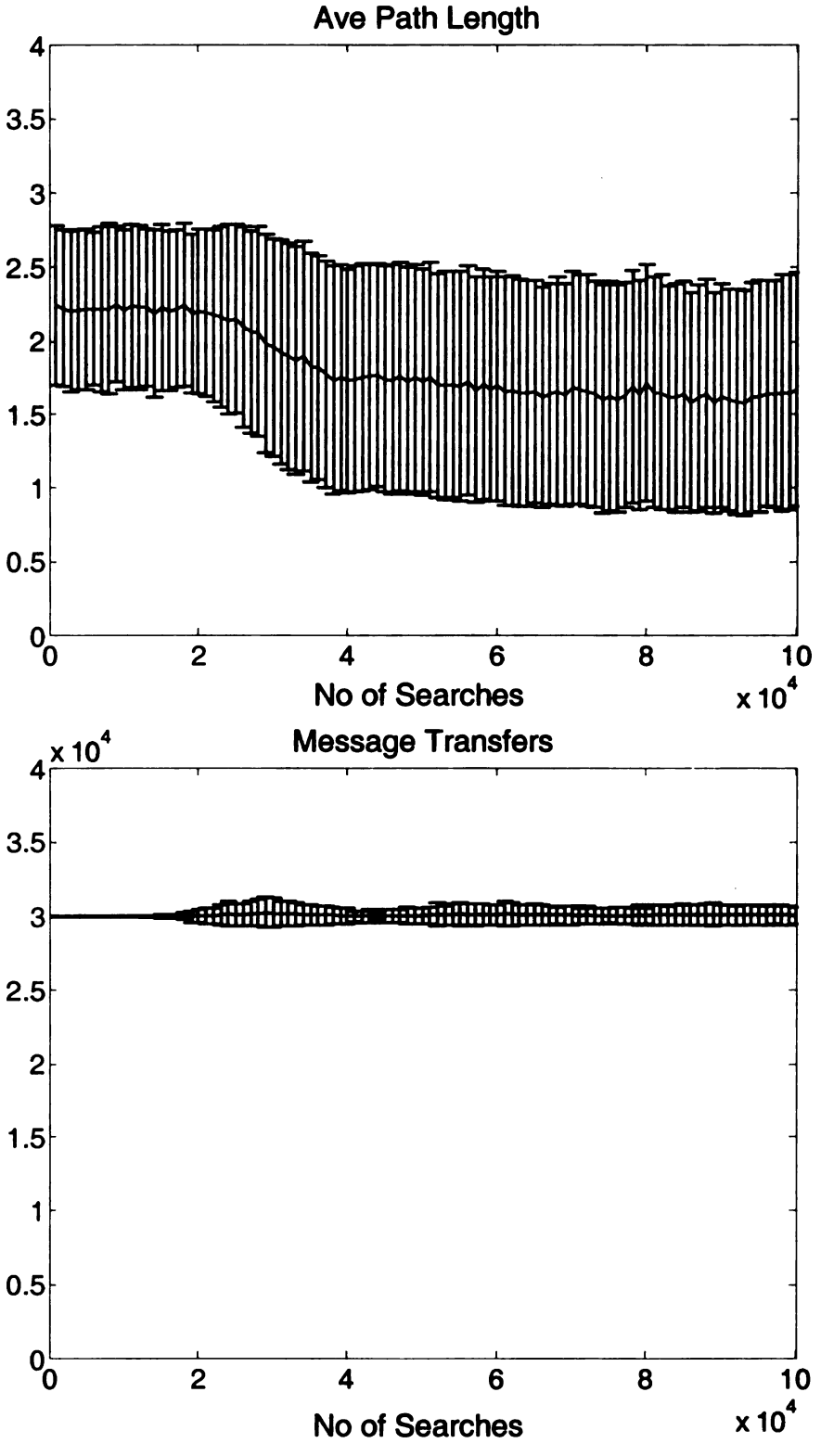

Figure 14: The 100000 searches in a 1000 node network, using DIAS routing in polling fashion for a Zipf distribution of documents over nodes.

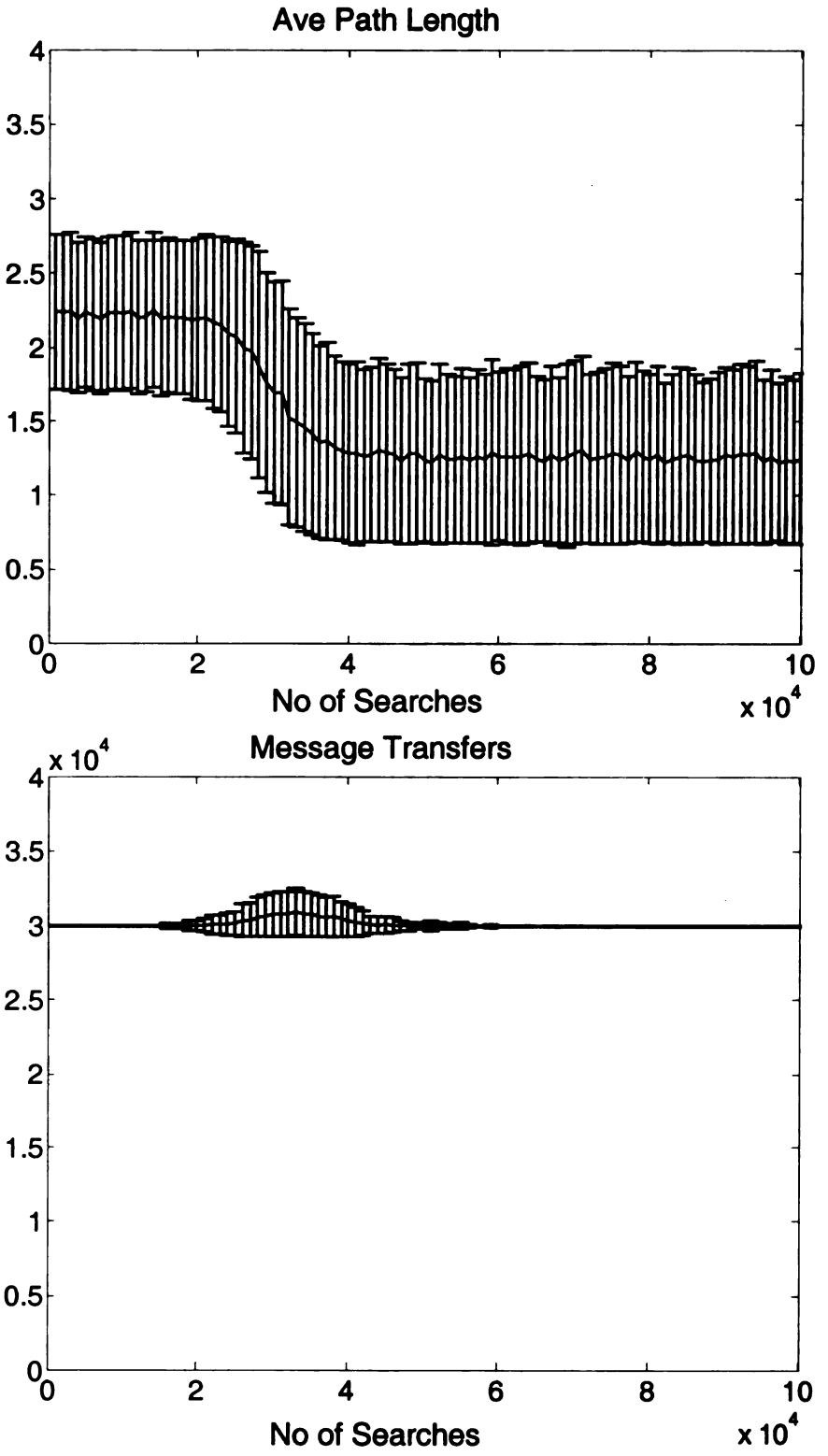

Figure 15: The 100000 searches in a 1000 node network, using DIAS routing in reporting fashion for a Zipf distribution of documents over nodes.

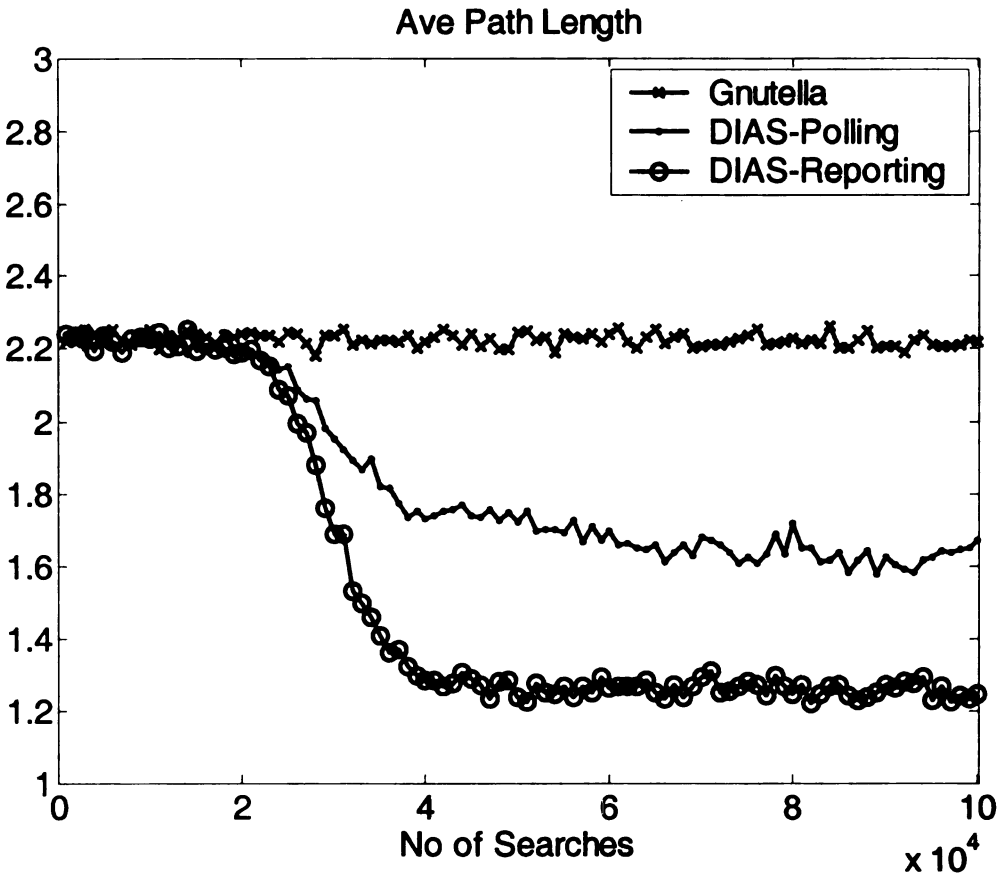

Figure 16: Gnutella vs. DIAS: Average Path Length

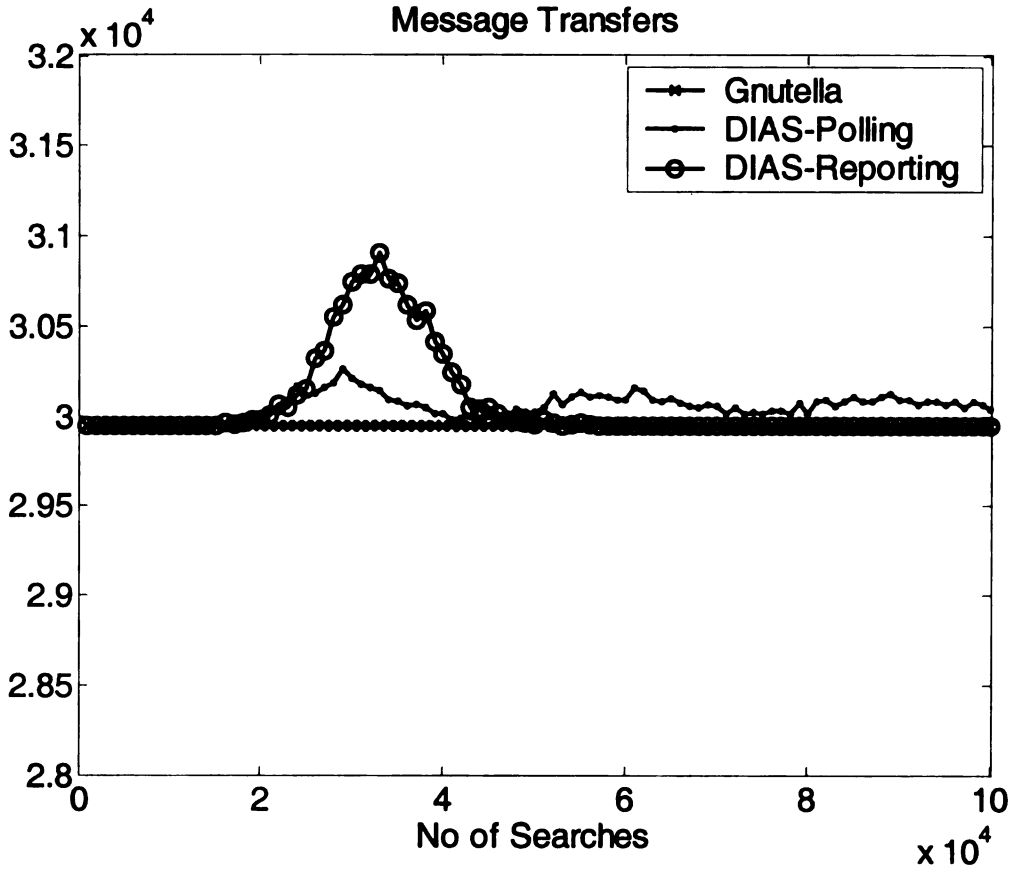

Figure 17: Gnutella vs. DIAS: traffic overhead

3.4.2 Different Configurations in DIAS of Polling Fashion

Parameters in the properties file may affect the performance of our protocol. Now let's take a look at the variation of performance when the parameters are set to different values in the polling fashion.

First, we set the value of the parameter NO_GROUPS to 200, the value of THRESHOLD to 0.4, and the value of BUDDY_UPDATE_LOOP is set to 20, 30, and 40 respectively. Figure 18 and Figure 19 present the simulation results when the parameter BUDDY_UPDATE_LOOP varied.

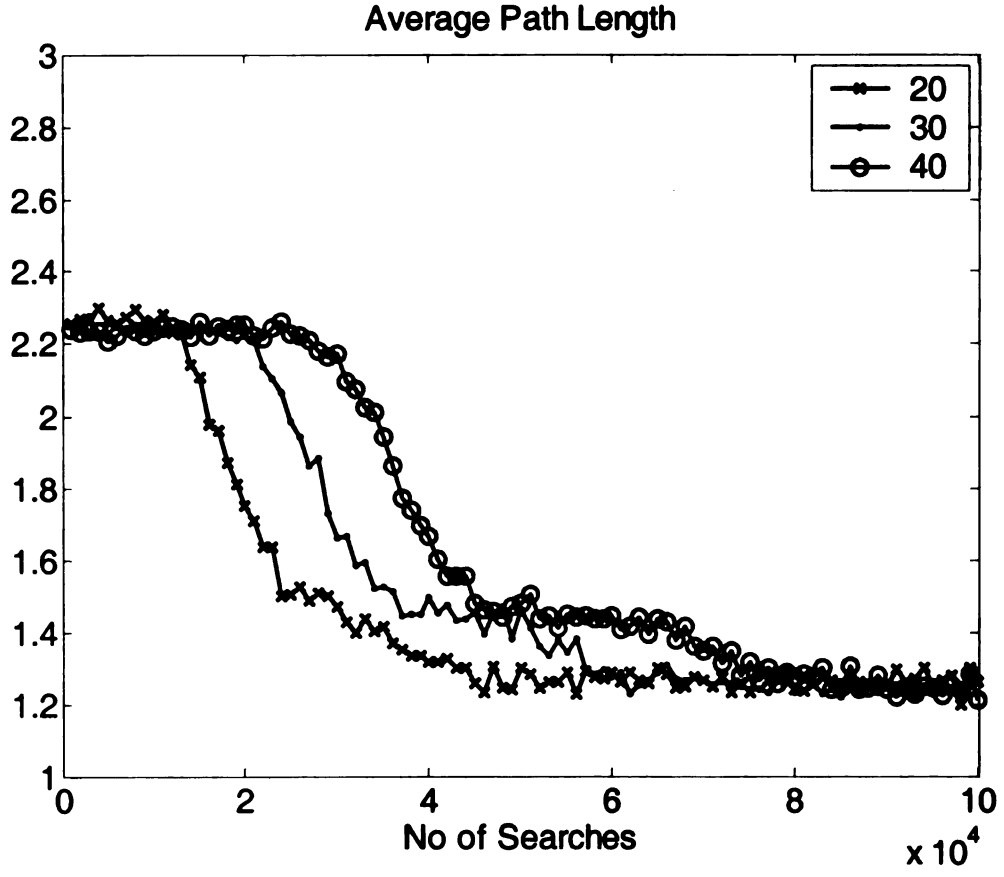

Figure 18: Variation of average path length when BUDDY_UPDATE_LOOP varied in the polling fashion

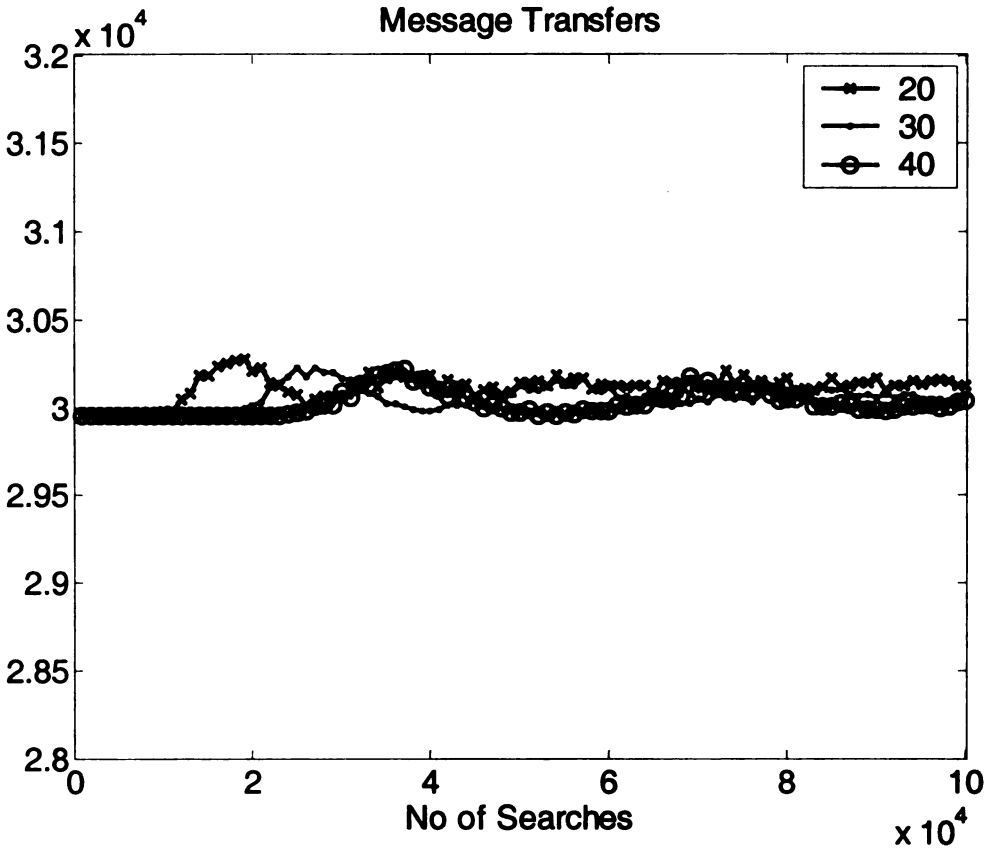

Figure 19: Variation of traffic overhead when BUDDY_UPDATE_LOOP varied in the polling fashion

The data in Figure 18 shows that the smaller the value of BUDDY_UPDATE_LOOP was, the earlier the average path length began to decrease. This indicates that if nodes in the DIAS system updated their buddy lists in shorter rounds, the time of warm-up in the system would be reduced, that is, the interest groups were generated more quickly. We know that when BUDDY_UPDATE_LOOP was wet to a smaller value, nodes in the system should update their buddy list more frequently. This may cause more traffic, and the additional overhead should be grander. But Figure 19 shows that there was no big difference among these three runs when the parameter BUDDY_UPDATE_LOOP varied.

Why? Because the additional traffic that the updating of buddy lists caused was so small that can be ignored.

Now let's take a look at the variation of the simulation result when the parameter THRESHOLD varied. This time we set the value of BUDDY_UPDATE_LOOP to 30, the value of THRESHOLD to 0.4, 0.5, and 0.6 respectively, and the value of NO_GROUPS is remained 200. Figure 20 and Figure 21 present the simulation results when THRESHOLD varied.

The data in Figure 20 shows that there was on big difference in average path length when the threshold varied. The reason is because nodes in our simulation shared similar interests in a high degree. So after a period of warm-up time, nodes sharing the common interest set were mostly clustered together. Thus the difference of threshold did not put much affection on the search efficiency. And because the additional traffic that the updating of buddy lists caused was so small that can be ignored, there was no big difference in traffic when the threshold varied either in Figure 21.

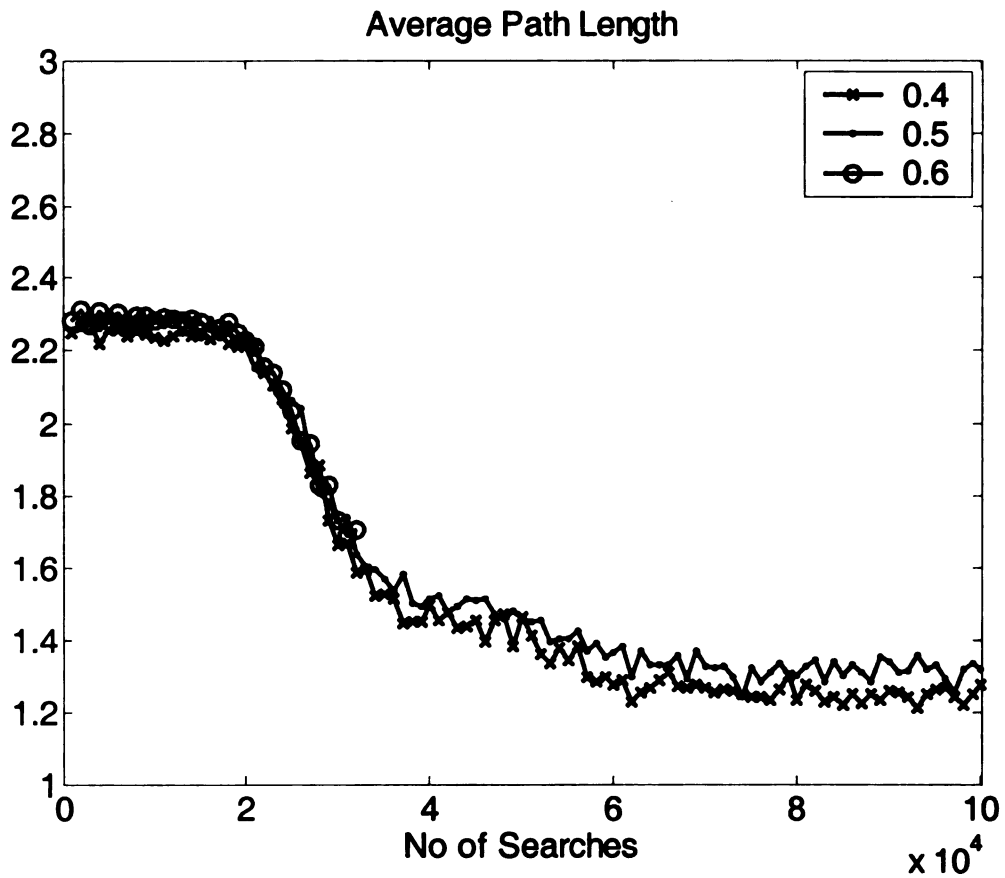

Figure 20: Variation of average path length when THRESHOLD varied in the polling fashion

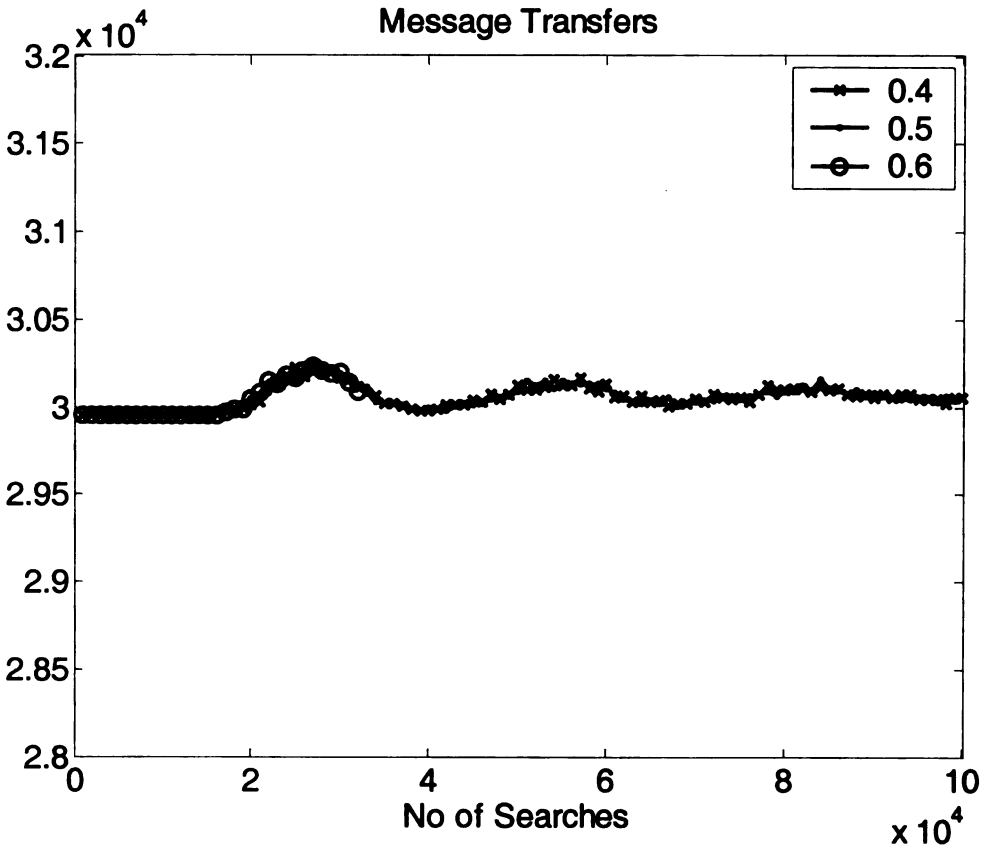

Figure 21: Variation of traffic when THRESHOLD varied in the polling fashion

3.4.3 Different Configurations in DIAS of Reporting Fashion

In this section, we will show the simulation result when parameters varied in the reporting fashion.

First, we set the value of the NO_GROUPS to 50, the value of NO_CONN_CHANGES to 5. Figure 22 and Figure 23 present the simulation result when the parameter NO_BUD_CHANGES was set to 5, 10, and 15, respectively.

The data in Figure 22 shows that the smaller the value of NO_BUD_CHANGES was, the more the average path length was reduced. That is because the requirement for a node to update its buddy list was easier to meet, nodes could update more frequently during the

warm-up time at the very beginning. So the interest groups were generated in a higher level. This caused the searching efficiency improved more. At the same time, the more updating was conducted, the more traffic was caused in the warm-up period. Thus, in Figure 23, the peak of the curve for NO_BUD_CHANGES 5 is the highest, and the peak of the curve for NO_BUD_CHANGES 15 is the lowest. And the additional traffic in the warm-up period was temporary, after that, the traffic for all the values of NO_BUD_CHANGES was equal to that in Gnutella network, because the interest groups were generated and stable.

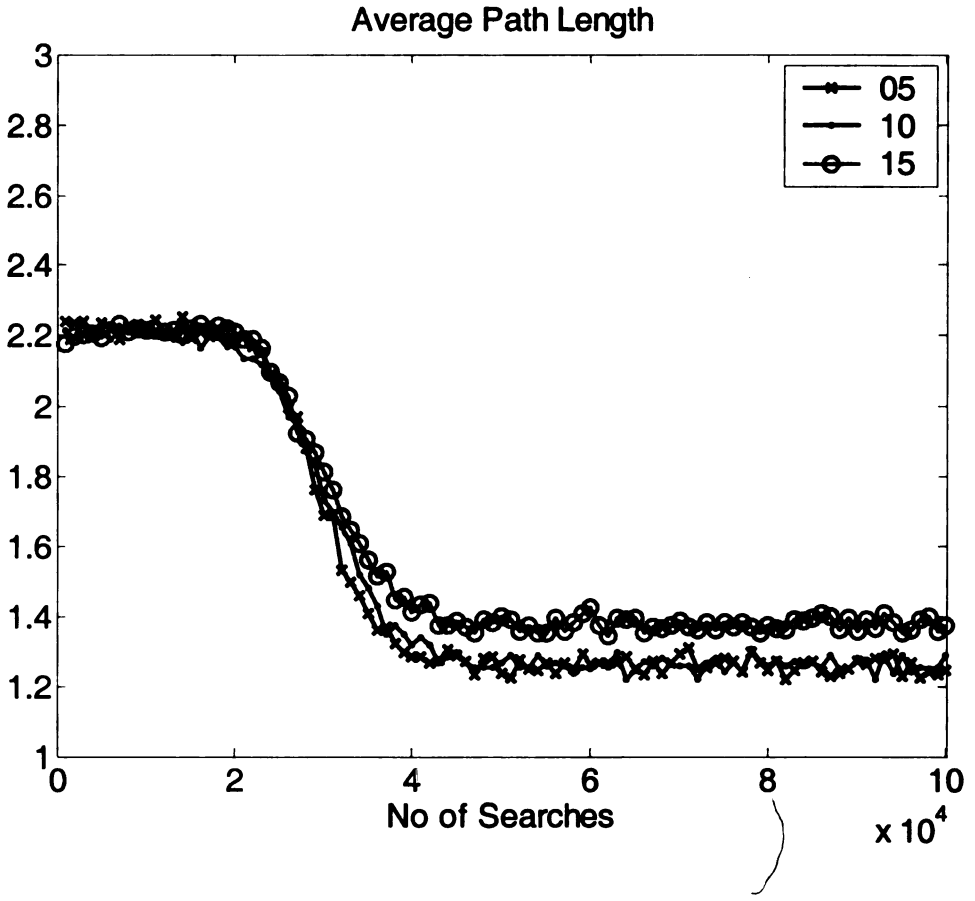

Figure 22: Variation of average path length when NO_BUD_CHANGES varied in the reporting fashion

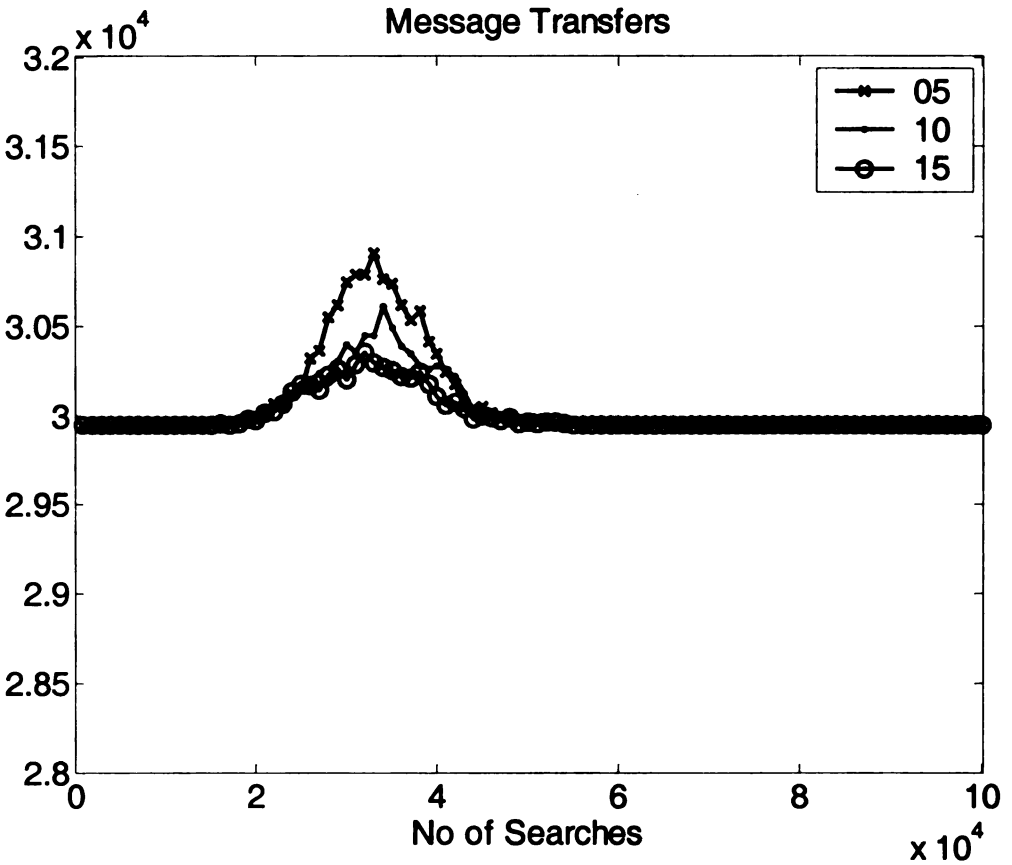

Figure 23: Variation of traffic when NO_BUD_CHANGES varied in the reporting fashion

Now, let's take a look at the variation of the simulation result when the parameter NO_CONN_CHANGES varied. This time we set the value of NO_BUD_CHANGES to 5, the value of NO_CONN_CHANGES to 5, 10, and 15, respectively, and the value of all the other parameters remained the same as in the simulation above.

The simulation results of this time shown in Figure 24 and Figure 25 is similar to the simulation results shown in Figure 22 and Figure 23 when the parameter NO_BUD_CHANGES varied. The data in Figure 24 shows that the smaller the value of NO_CONN_CHANGES was, the more the average path length was reduced. The reason is that the requirement for a node to report its variation was easier to meet, thus during

the same period, more nodes could get enough varying neighbors to update their buddy lists. As a result, buddy lists were updated more frequently during the warm-up time. So the interest groups were generated in a higher level, which caused the searching efficiency improved more. Also as described above, the more updating was conducted, the more traffic was caused in the warm-up period. Thus, in Figure 25, the peak of the curve for NO_CONN_CHANGES 5 is the highest, and the peak of the curve for NO_CONN_CHANGES 15 is the lowest. And the additional traffic in the warm-up period was temporary, after that, the traffic for all the values of NO_CONN_CHANGES was equal to that in Gnutella network, because the interest groups were generated and stable.

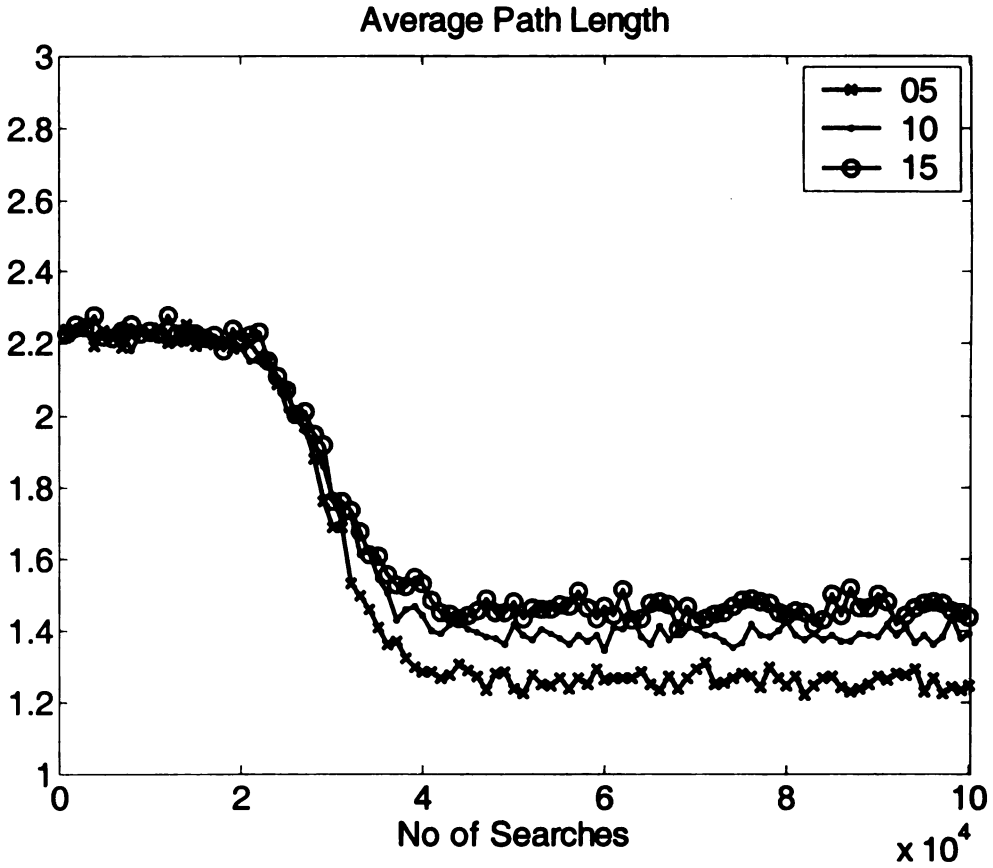

Figure 24: Variation of average path length when NO_CONN_CHANGES varied in the reporting fashion

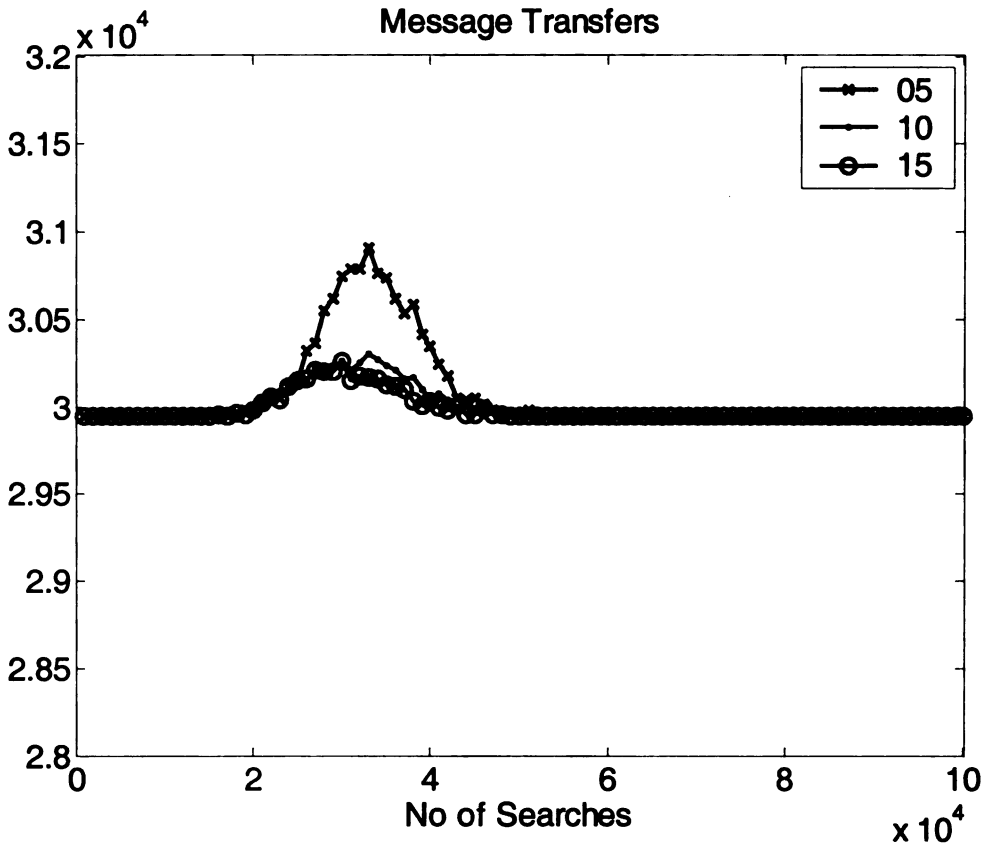

Figure 25: Variation of traffic overhead when NO_CONN_CHANGES varied in the Reporting Fashion

CHAPTER 4 DIAS IMPLEMENTATION ON JXTA

4.1 JXTA Architecture

We have introduced JXTA briefly in Chapter 1. We know that project JXTA is conceived with three main objectives intended to address the shortcoming of the P2P system: interoperability, platform independence, and ubiquity. Distributed computing applications built on top of JXTA can across different P2P systems and communities. Language independence, system independence and network independence can be achieved. Every device in these applications, wire or wireless, has a digital heartbeat.

In the JXTA network, each peer resource is associated with a single unique peer ID. For a given peer, all of the available network interface address is encapsulated for it by a peer endpoint, so this peer can only be addressed by its peer ID. Others peers can make choice among the list of available peer endpoint addresses to achieve the most efficient communication with this peer while getting the advertisement from the peer endpoint, here "advertisements" are XML documents that peers cache, publish and exchange to explore and discover network resources. The notion of interest base is also applied in JXTA project. Peers in JXTA network self-organize into groups based on their interest. If a set of peers have a common set of interests, and have agreed on a common set of policies, for example, membership, then they can establish a group [11]. Each group is also associated with a unique group ID.

For developers, Project JXTA provides a set of building blocks that provide a solid foundation for distributed applications. Also these blocks can support the common functions required by any P2P system. Also we know that distributed computing applications require a great deal of infrastructure, and building this framework is typically time consuming and resource-intensive. By using Project JXTA, we can focus on creating innovative software applications, and do not need to care about the infrastructure of P2P system [19]. Therefore we select JXTA as the underlying infrastructure for our DIAS application. Figure 26 shows the JXTA implementation overview:

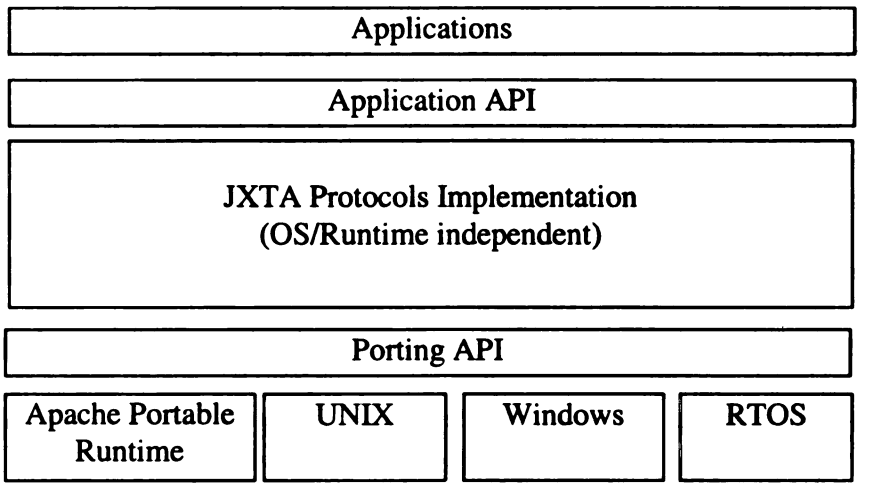

Figure 26: JXTA application overview

InstantP2P is a demo application that illustrates key concepts of Project JXTA and P2P system. It is consist of two main applications: Chat application and File sharing application. The InstantP2P application provides functionality for secure one-to-one chat, group chat, and sharing, searching, downloading documents within a peer group by using the JXTA platform core building blocks. It can create, discover, and join groups, create a

connection between two peers in the one-to-one chat function, create connections among a group of peers in the group chat function, as well as operations on files.

4.2 The Applications in DIAS

In this section, we describe how our DIAS is designed on top of the InstantP2P application.

4.2.1 Overview

A started InstantP2p application represents a peer in a P2P network. The name of the peer is set as soon as the initial entry for the peer is created. After boots-trapping the start-up of the JXTA platform, the GUI application will instantiate one instance of the class *net.jxta.instantp2p.PeerGroupManager*. In order to implement our DIAS system, most of our modification was focus on this class, because it is the heart of the application. *PeerGroupManager* is responsible for monitoring and influencing the buddy discoveries. It generates and maintains the buddy list for a peer using the algorithm we describe in Chapter 3. Upon instantiation it will immediately trigger a remote search of neighbors in the P2P network.

The GUI application can retrieve a list of remote peers using the following methods:

- *public Enumeration getNeighborList()*: Returns an enumeration of all the neighbors of this peer. The objects in this enumeration are net.jxta.protocol.PeerGroupAdvertisement objects.
- *public Enumeration getBuddyList()*: Returns an enumeration of all the buddies of this peer.

The *PeerGroupManager* class might also want to search for peers with specific names only. It can do so by using the following method:

- *public void searchPeers (String peerName)*: This will trigger the remote discovery of peers. If the parameter *peerName* is not null, the remote search is restricted to peers with the specified name.

The above methods do not return a list of discovered peers, since this happens asynchronously. Instead interested classes can get the search result by implementing the interface *net.jxta.instantp2p.GroupStructureListener* via the methods in this interface. The classes can receive the notifications by calling the method *public void addStructureListener (roupStructureListener listener) to* register, and terminate the notifications by calling the method *public void removeStructureListener (GroupStructureListener listener).*

In order to be informed if new peers are discovered, classes can implement the *net.jxta.instantp2p.GroupStructureListener* interface via the method *public void peerStructureChanged ()*. To retrieve a list of peers, classes can call the method *getPeers*() in the class *net.jxta.instantp2p.PeerGroupManager*. In order to be informed if the currently selected peer changed, classes can implement the *net.jxta.instantp2p.PeerListener* interface via the method *public void peerChanged (PeerGroupAdvertisement pgAdv)*

The *PeerGroupSearcher* class is another major class we modified to implement our system. It also has a notion of a currently selected peer. The GUI application is responsible to inform the *PeerGroupSearcher* instance of changes in the currently selected peer using the method *public void selectedPeerChanged (PeerGroupAdvertisement adv).*

PeerGroupManager can also communicate the change of the peer in one part of the GUI to the other parts of the GUI. Classes interested in the changes of the selected peer implement the *net.jxta.instantp2p.PeerListener* interface and will then be informed via the methods in that interface of the changes. To receive the notifications interested classes register themselves by using the method *addPeerListener (PeerListener listener).* Notification can be terminated by calling the method *removePeerListener (PeerListener listener).* The currently selected peer can be retrieved via the method *PeerGroupAdvertisement getSelectedPeer().*

4.2.2 Chat Application

The chat application allows a peer to chat with its neighbors or its buddies. The functions are implemented in the class *net.jxta.instantp2p.Chat* and the interface *net.jxta.instantp2p.MessageBoard* that we modified to apply to the chat application in our system.

The class *net.jxta.instantp2p.Chat* establishes a chat between a peer and its neighbor or its buddy. Once a peer wants to chat, this class will publish the appropriate advertisement remotely by the method *login*. The name used for the user is its peer name returned by the method *getMyPeerName()* in the class *PeerGroupManager*. In order for the peer to receive message, an input pipe is established and monitored also by the *login* method in the *Chat* class. If the peer sends a message, the message will be sent to the currently selected peer as reported by the class *PeerGroupManager* via an output pipe created and monitored by the *login* method.

The user is informed about the success or failure of the attempt to open a chat via calls to the *MessageBoard* instance. If there is no currently selected peer to chat with, the peer is instructed to select a peer. If a peer wants to chat with its buddy, then a secure chat is initiated, i.e. all information sent and received between the chatting buddies is encrypted. When an output pipe to a selected buddy is opened, a public key will be send to the corresponding buddy.

The constructor of this class *public Chat (PeerGroupSearcher manager, MessageBoard messageBoard) can* register itself as a PeerListener to the *manager* instance, which allows the chat application to access the current neighbors of the peer. The instance *messageBoard* of the *MessageBoard* class is used to display the received messages to the peer. The method *public synchronized void logout ()* terminates the current chat. The monitoring of the input pipe is suspended. The method *public void sendMessageToPeers (String message)* propagates the message to the output pipe and thus send to the neighbor. If the chat channel is secure, the message is encrypted before it is sent to the buddy.

Messages may be of different types and received by different neighbors. Thus, a GUI application would probably render these messages in different colors. In addition there is a method that requests information about whether the user wants to accept certain chat proposals.

4.2.3 File Application

The file sharing application allows a peer to manage its local content, search remote content from the peers in the P2P network, and maintain the connections between its neighbors and itself. These functions are implemented in several classes and interfaces.

Content distributed in our DIAS system is described by *net.jxta.share.ContentAdvertisement* object. The *ContentAdvertisement* class extends the JXTA *Advertisement* object and thus allows propagation along normal JXTA discovery channels. The methods below return information about the content described by the *ContentAdvertisement* object:

- *public abstract String getName()*: Returns a name that identifies the content. This is often the file name of the file described by the *ContentAdvertisement* object.
- *public abstract ContentId getContentId()*: Returns the UID that uniquely identifies the content described in the *ContentAdvertisement* object.
- *public abstract long getLength()*: Returns the length of the content.
- *public abstract String getType()*: Returns the mime type of the content in order to select the correct method to display the content once retrieved.
- *public abstract String getDescription()*: Returns a detailed description of the content.

Local content is described by *net.jxta.share.Content* object. The method in this interface *public ContentAdvertisement getContentAdvertisement()* allows to extract detailed information about the local content by return a *ContentAdvertisement* instant. Remote content is described by *net.jxta.instantp2p.SearchResult* object. The method in this inter-

face *public ContentAdvertisement getContentAdvertisement()* allows to retrieve a description about the remote content.

The class *net.jxta.instantp2p.SearchManager* is the interface to the file searching application. Its constructor *public SearchManager(PeerGroupSearcher manager)* retrieves peer information from the parameter *manager*. This class registers itself as a PeerListener to *manager*. Some major methods in this class are described below:

- *public void startSearch(String searchString):* Starts an asynchronous search for remote data. The result is communicated to the registered *ContentListener* objects.
- *public void addContent(String path) & public void addContent(File addFiles[]):* Allows a peer to add content locally.
- *public void removeContent(Enumeration enum) & public void removeContent(Content c):* Allows a peer to remove its local content.

If a peer wants to download the data from a remote peer, the method *retrieveContent()* in the class *net.jxta.instantp2p.SearchManager* can be used to retrieve the data via the *net.jxta.instantp2p.ContentListener* interface. To retrieve the actual data, an input pipe needs to be opened to the peer in which the actual data resides. Since the download process may be time consuming, it is done asynchronously. The method *public void finishedRetrieve(String url)* is called once the retrieval has succeeded.

The class *net.jxta.instantp2p.ConnectionManager* is responsible for connection maintenance. Some major methods in this class are described below:

- *public void addConnection(String peerName):* Allows a peer to add connection between a remote peer and itself.

- *public void removeContent(String peerName):* Allows a peer to remove connection between its neighbor and itself.

The classes above are the major classes we modified for the file application of our system.

4.3 Implementation

In this section, we describe how the DIAS application is built and work.

4.3.1 Recommended System Requirements

- Any Java enabled platforms: Windows (95, 98, ME, 2000, NT, XP), Solaris, Linux, Unix, MaxOSX.
- 200MHz Pentium class computer with 128MB of RAM, 5 MB free disk space.

4.3.2 Building Requirements

- Windows NT Version 4 Service Pack 3, Windows 2000/98/ME
- Java 2 JDK 1.3.1
- Cygwin: a Linux-like environment for Windows
- Ant: A Java-based build tool in Apache.

4.3.3 Running DIAS

- **Configuration**

The first time when DIAS is run, the JXTA Configuration window that is shown in Figure 27 will appear.

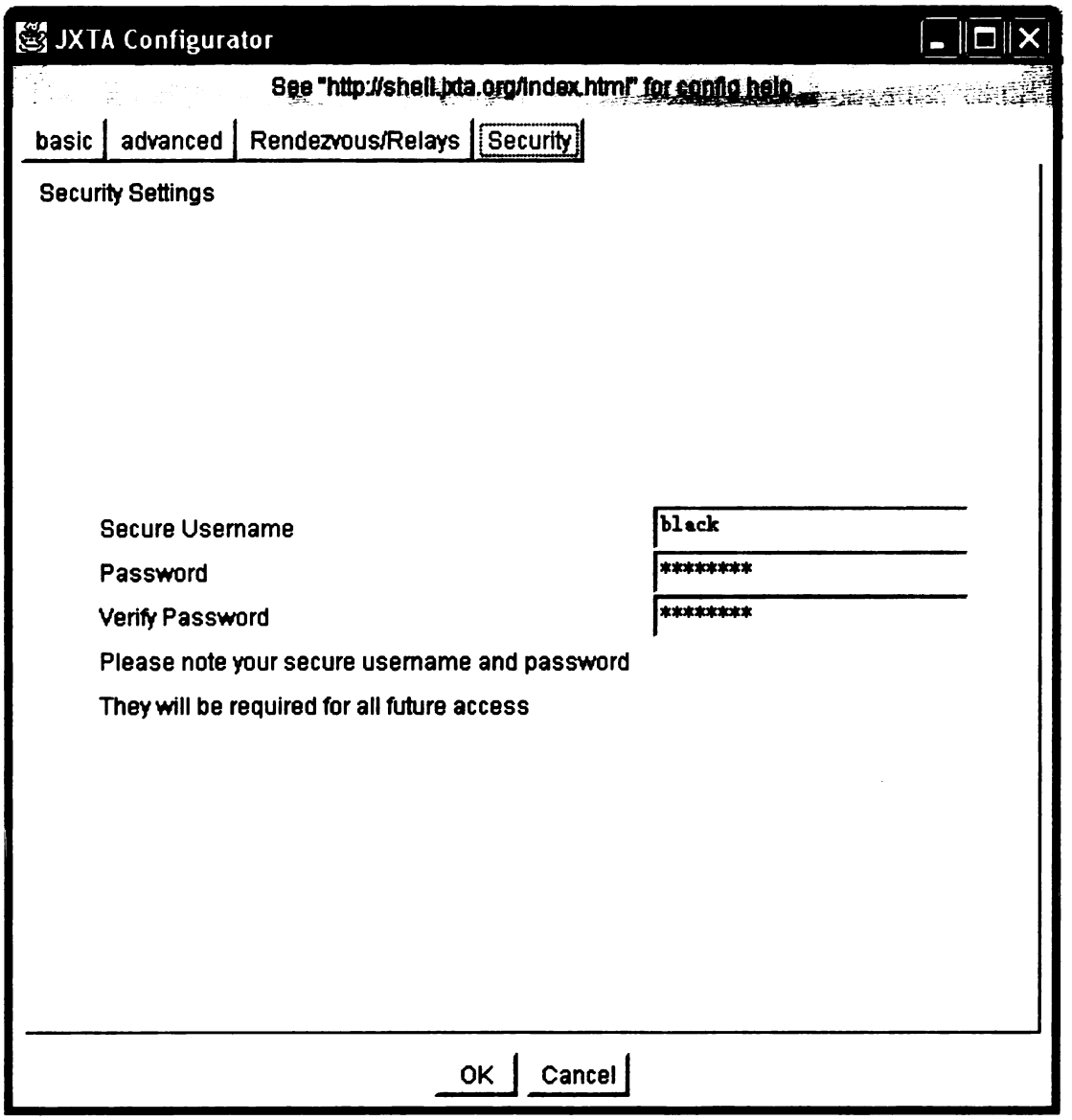

Figure 27: JXTA configuration window

In this window, a user is required to input some necessary information such as "Peer Name" in the "basic" tab, "Secure Username" and "Password" in the "Security" tab. After the initial setup, a user is only required to enter a name and password in the Secure Login window shown in Figure 28 to connect to the JXTA network. Each time the DIAS

is restarted, the user will have to verify this Secure Username identity, because it is his/her unique identity on the JXTA Network.

Figure 28: Login window

When the application launches, the main window of the DIAS application will come out. This window has menu options that control key functionality of the system and a panel contains four tabs: Secure Chat, Local Content, Search, and Maintain Connections. The first tab corresponds to the Chat application. The last three tabs consist of the File application.

- **Secure Chat**

The search tab is shown in Figure 29. In the left half of this tab, the buddies and neighbors of this peer are listed. The user can select a certain neighbor to open a chat or a buddy to open a secure chat. The text box in the bottom of the tab is used for the user to write and send message. The sent and received messages are displayed in the right half of the tab. The messages sent to and received from different neighbors of buddies are displayed in different color. That is shown in Figure 30.

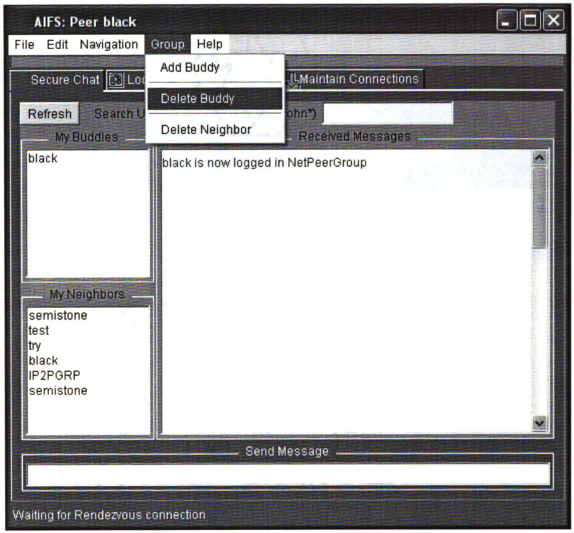

Figure 29: Secure chat tab

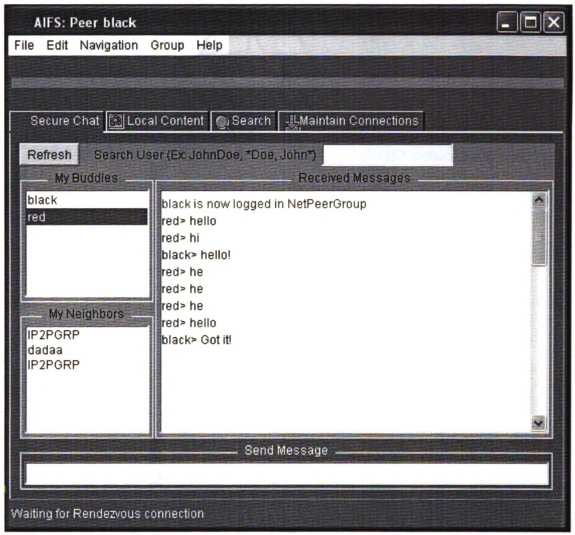

Figure 30: The chat between two buddies

- **Local Content Management**

Figure 31 shows the local content tab. The user can use this tab to manage his/her local content. The indexes of content are listed in the left half of the tab and the detailed description of content are displayed in the right half. The user can add a new piece of content by click the "Add" button, and delete an undesired piece of content by highlight the index of this piece of content and click the "Delete" button.

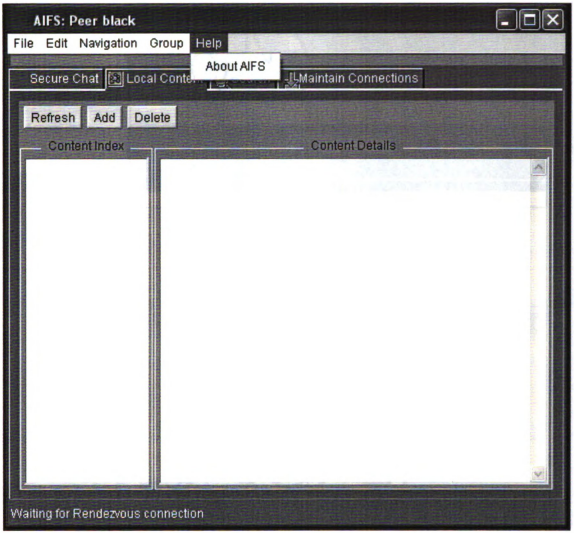

Figure 31: Local content tab

- **Search Function**

The search tab is shown in Figure 32. When the user wants to search a certain piece of content, he/she should first input the keywords he/she wants to search, then click the "Search" button. When the search succeeds, all the matched content will be displayed in the bottom box. The information includes the ID, the name, and the owner peer of each piece of content. After the user has conducted several searches, he/she clear all the pervious search results by clicking the "Clear" button.

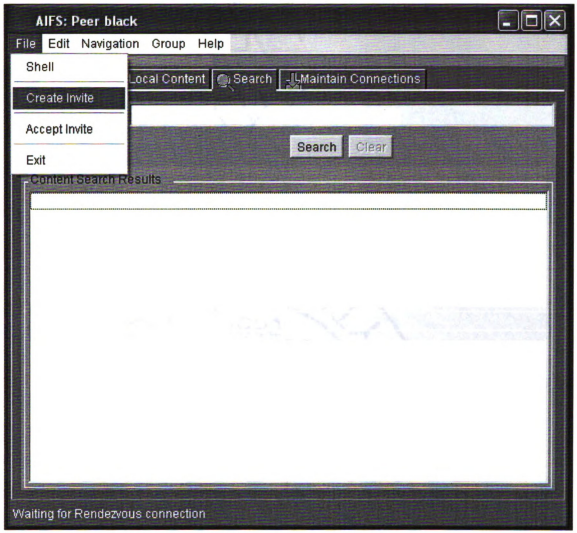

Figure 32: Search tab

- **Connection Maintenance**

Figure 33 shows the maintain connection tab. The peer ID and peer name of the user's buddies and neighbors are listed in the two boxes respectively. Though the connections of each peer are maintained automatically using the algorithm we described in Chapter 3, a user can manually delete a connection to its buddy or neighbor by clicking the "Add Connection" or add a connection by clicking the "Remove Connection" button to look for the available peers in the network according to some special willingness.

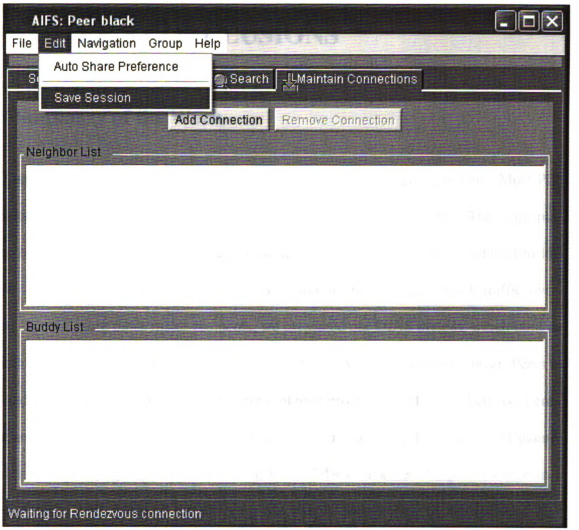

Figure 33: Maintain connection tab

CHAPTER 5 CONCLUSIONS

5.1 Summary of the Work

Content location in decentralized P2P networks is a challenging problem. Most P2P protocols, such as Gnutella, employ fairly inefficient search approaches. The main purpose of our research is to find out a scheme that can make use of peer's interest to improve the search efficiency in P2P networks without producing too much traffic overhead. By analyzing those major protocols, we presented our DIAS system, which built its search scheme based on the conception of interest retrieval in the semantic layer. Peers in a DIAS network are clustered into different interest groups according to their local content and historical searching behaviors. And they issue their search requirements mainly within the interest groups associated with them. DIAS enhances the performance in a considerable level, without complicating the simple architecture of unstructured P2P networks.

The simulation results show that using our scheme in the P2P environment, including ad hoc networks, the searching performance will be optimized dramatically. Also we implemented DIAS on top of the JXTA application. Two applications are implemented in the system: Chat application and File application.

Our main contributions are:

- Propose a P2P search approach which generates and maintains interest groups by computing interest similarity between peers

- Show the improvement of search efficiency with moderate traffic overhead by simulation
- Demonstrate the feasibility of this approach by the implementation on JXTA.

5.2 Future Work

By the simulation result and the implementation on JXTA, we can conclude that our AIFS system can support a more efficient search process without producing additional overhead. Also this scheme is feasible in real network system. But as we have mentioned above, it cannot reduce traffic overhead as in the NeuroGrid system. Our AIFS protocol still employs the query flooding approach in the traditional Gnutella system. So one of our future works is to try to reduce the traffic overhead of the system. We may accomplish this goal by avoiding the random flooding fashion. A possible solution is to utilize the rank of each buddy and neighbor of a peer and only forward the querying message to the limited number of buddies with the highest rank. A query from a node will be sent to the first several (group leader index) buddies in the interest group corresponding to the keywords in a given query.

When a search succeeds, a peer will download its wanted content from the remote peer using the traditional scheme in Gnutella and other P2P networks. In this scheme, only one connection will be established between the querying peer and the remote destination peer. Thus the download speed is really low and the efficiency is not satisfactory. So to optimize the download pattern is another critical task in our future work. The strategy used in Kazaa gives us some hint. We may utilize the idea in Kazaa to establish several connections between the querying peer and all the remote peers that have the wanted content.

Also the duplication of the searched content in each peer along the propagation path may be employed.

BIBLIOGRAPHY

[1] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An Analysis of Internet Content Delivery Systems," In *Proceedings of OSDI 2002*, 2002.

[2] S. Joseph, "Adaptive Routing in Distributed Decentralized Systems: NeuroGrid, Gnutella and Freenet," In *Proceedings of workshop on Infrastructure for Agents, MAS, and Scalable MAS, Autonomous Agents 2001*, 2001.

[3] Y. Fu, L. Cherkasova, W. Tang, and A. Vahdat, "EtE: Passive End-to-End Internet Service Performance Monitoring," In *Proceedings of USENIX 2002 Annual Conference*, 2002.

[4] E. Cohen, A. Fiat, and H. Kaplan, "Associative search in peerto -peer networks: Harnessing latent semantics," In *Proceedings of IEEE INFOCOM 2003*, 2003.

[5] C. Tang, Z. Xu, and M. Mahalingam, "Peersearch: Efficient information retrieval in peer-to-peer networks," HP Labs, Technical Report HPL-2002-198, 2002.

[6] S. Joseph, "NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks," In *Proceedings of the International Workshop on Peer-to-Peer Computing (co-located with Networking 2002)*, 2002.

[7] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," In *Proceedings of INFOCOM 2003*, 2003.

[8] J. Xiao, Y. Zhang, X. Jia, and T. Li, "Measuring Similarity of Interests for Clustering Web-Users," In *Proceedings of Australiasian Database Conference*, Gold Coast, Queensland, Australia, 2001.

[9] 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 ACM Interna-tional Conference on Supercomputing*, 2002.

[10] C. J. v. R. B.Sc., P. D. Dip. NAAC, M.B.C.S., F.I.E.E., C.Eng., and F.R.S.E., *INFORMATION RETRIEVAL*, 2nd ed. London: Butterworths, 1979.

[11] B. Traversat, M. Abdelaziz, D. Doolin, M. Duigou, J.-C. Hugly, and Eric Pouyoul, "Project JXTA-C: Enabling a Web of Things," In *Proceedings of 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, Big Island, Hawaii, 2003.

[12] JXTA for J2ME: Extending the Reach of Wireless With JXTA Technology, http://www.jxta.org/project/www/docs/JXTA4J2ME.pdf

[13] Project JXTA 2.0 Super-Peer Virtual Network, http://www.jxta.org/project/www/white_papers.html

[14] Z. D. Michael W. Berry, Elizabeth R. Jessup, "Matrices, Vector Spaces, and Information Retrieval," *Society for Industrial and Applied Mathematics*, vol. 41, pp. 335 - 362, 1999.

[15] J. W. Sons, *Information Retrieval Systems: Characteristics, Testing and Evaluation*, 2nd ed. New York: F. Wilfrid Lancaster, 1979.

[16] R. Baeza-Yates and B. Ribeiro-Neto, "Modern Information Retreival," *ACM SIGMETRICS Performance Evaluation Review*, 1999.

[17] P. Pathak, M. Gordon, and W. Fan, "Effective information retrieval using genetic algorithms based matching functions adaptation," In *Proceedings of the 33rd Hawaii International Conference on System Science, 2000*, Hawaii, 2000.

[18] Zipf, Power-laws, and Pareto - a ranking tutorial, http://ginger.hpl.hp.com/shl/papers/ranking/ranking.html

[19] Project JXTA Technology: Creating Connected Communities, http://www.jxta.org