2006

Exploiting semantic locality to improve peer-to-peer search mechanisms

Ajitabh Sharan

Exploiting Semantic Locality to Improve Peer-to-Peer Search Mechanisms

by

Ajitabh Sharan

A Thesis

Submitted to the Faculty

of the

ROCHESTER INSTITUTE OF TECHNOLOGY

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

August 2006

APPROVED:

Professor James Minseok Kwon, Thesis Advisor

Professor Hans-Peter Bischof, Reader

Professor Rajendra K Raj, Observer

Abstract

A Peer-to-Peer(P2P) network is the most popular technology in file sharing today. With the advent of various commercial and non-commercial applications like KaZaA, Gnutella, a P2P network has exercised its growth and popularity to the maximum. Every node (peer) in a P2P network acts as both a client and a server for other peers. A search in P2P network is performed as a query relayed between peers until the peer that contains the searched data is found. Huge data size, complex management requirements, dynamic network conditions and distributed systems are some of the difficult challenges a P2P system faces while performing a search. Moreover, a blind and uninformed search leads to performance degradation and wastage of resources. To address these weaknesses, techniques like Distributed Hash Table (DHT) has been proposed to place a tight constraint on the node placement. However, it does not considers semantic significance of the data.

We propose a new peer to peer search protocol that identifies locality in a P2P network to mitigate the complexity in data searching. Locality is a logical semantic categorization of a group of peers sharing common data. With the help of locality information, our search model offers more informed and intelligent search for different queries. To evaluate the effectiveness of our model we propose a new P2P search protocol - *LocalChord*. LocalChord relies on Chord and demonstrates potential of our proposed locality scheme by re-modelling Chord as a Chord of sub-chords.

Acknowledgements

I would like to begin with my mother Rupa Sharan for giving me courage and strength I needed to complete my goal. I would also like to express thanks to my father Ranjit Sharan for his continued support and trust.

I would like to thank my committee members for giving me the opportunity to perform. My deep appreciation goes to Prof Fereydoun Kazemian for his kindness to accept the position of Reader in last hours of this work. A special thanks to Prof Rajendra K Raj for his support. Words are inadequate to express my thanks to my advisor Prof. James Minseok Kwon for his expertise in what turned out to be an exciting quest for this work eventually finishing up as thesis. To him I owe the most overwhelming debt of gratitude.

Last but not the least, I would like to thank all my family and friends who directly or indirectly helped me in completing my Master in Computer Science. I praise Rochester Institute of Technology for providing the best education and preparing student to excel in every phase of life.

Contents

List of Figures

Chapter 1

Introduction

A Peer-to-Peer (P2P) system is a distributed system design for multiple hosts to share files and work in conjunction. In a P2P system, peers establish a direct communication with each other to transfer and share files without intervention of any intermediate hosts. The search for a peer in a P2P system is a challenge because of the random distribution of the nodes across the network. This can be described as analogous to a hypothetical society described below

Imagine a hypothetical society, where a person can communicate with other people only if they are physically reachable to him. If a person wants to find a piece of information, then merely knowing people within his reach will not guarantee him the information. How can a person find his information then? One possible way is to ask every other person within his vicinity for the information. People in his vicinity can then guide him to other people who may have the information.

Common problems in such a society is the lack of information specific to a person's query. Moreover, a person has no prior knowledge of the domains where he can target his search. Peer-to-Peer system is analogous to this model of society and shares the same search problem. To mitigate the problem of searching, structured

models were proposed like Chord. Chord creates an overlay (ring) on top of a P2P system and facilitates in realizing an efficient search mechanism but at the same time fails to address the issue specific to the semantics of a query. We propose a new P2P search scheme - "L-Chord", in which a query adapts its path defined by its semantics. Different localities (overlays) are created on top of the network to group data, sharing the same semantics. The primary motivation is to target the search queries over localities which are specific to their semantics.

Local Chord (L-Chord) is a new P2P search model built on the top of Chord [1]. L-Chord exploits the semantics of the search and attempts to target the query over a domain containing data with similar semantics. L-Chord re-models Chord as "Chord of Chords". Each sub-chord is defined as a locality attributed to the semantics of the queries. L-Chord is scalable and fault-tolerant to the abnormal departure of nodes. Moreover, L-Chord uses DHT specifics and utilizes Chord's (key,value) paradigm. Our results demonstrate, with small extra overhead a better P2P search scheme can be realized. We have shown that by keeping a control relationship between the size and number of localities the performance can be optimized for better performance and fewer hops.

The rest of the discussion begins by building bits and pieces necessary to understand the search problem in a P2P system by discussing previous work in Chapter 2. Chapter 2 also provides an insight in to different P2P systems and draws classifications on the basis of their network model and peer-to-peer nature. Chapter 3 defines the problem statement followed by the introduction of the proposed generic framework solution to the problem. Chapter 4 describes the application of the framework - L-Chord a semantic based P2P search scheme. In Chapter 5, results were discussed and the performance of L-Chord is evaluated. Finally, Chapter 6 summarizes the proposed scheme and ends with a discussion for future enhancements and extensions.

Chapter 2

Related Work

Computer Networks can be categorized in to two different types on the basis of the roles played by a participating computer - 1) Client-server model and 2) Distributed system model. In a client-server model, one node plays the role of a server providing services to other hosts who act as clients. On the other hand, distributed systems are often described in the notion of several hosts working in conjunction to complete a task. Unlike a client-server network, participants of a distributed system are independent with equal capabilities. This means a node can play the roles of both a server and a client. A peer-to-peer (P2P) system evolves out of a distributed system model. In this chapter we will provide an overview of P2P [2, 3] networks. Section 2.3 discusses about different P2P systems and their search mechanism.

2.1 What is a Peer-to-Peer network?

A peer-to-peer [3] network is a distributed system architecture in which every peer (host) communicates with other peers without the intervention of intermediate hosts. Every peer in a P2P system has similar computing capabilities and they share

resources to work in conjunction. The functionality and the integrity of the system remains unaffected even if a peer fails.

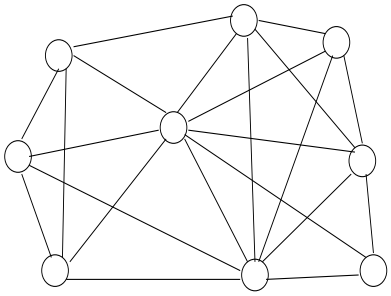


Figure 2.1: A Peer-to-Peer network. Notice there is no central server

The absence of a central server uniquely distinguishes P2P network from a client-server model. The most common applications of P2P networks are file sharing, distributed file storage, resource sharing and distributed computing. Schollmeier [4] provides a good survey and discussion on different forms of accepted P2P definitions and its applications. In upcoming sections we have provided a discussion on different flavors of P2P networks by describing a P2P application commercially and academically released so far. Following section describes different types of network which shares a close co-relation with P2P networks often known as "Overlay Networks".

2.2 Overlay Networks

Overlay networks model a network by creating a logical path of nodes on top of it. The overlays facilitate in routing and searching mechanism. Doval [5] describes overlay networks as a virtual topology above the transport protocol. Researchers have exploited overlay networks to model P2P networks. For example, Content Addressable Networks (CAN) [6] build overlay networks on top of a P2P network by creating a virtual d-dimensional cartesian coordinate space. Each node in CAN

belongs to a zone and data is stored as a (key, value) pair at any point on the logical coordinate space.

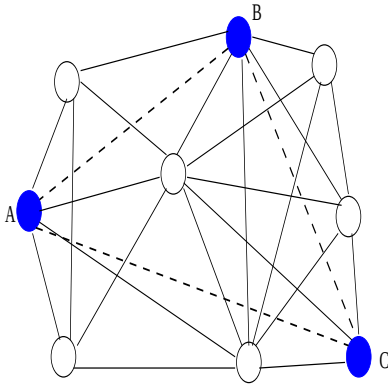


Figure 2.2: An overlay network on top of a Peer-to-Peer network. Nodes A, B and C forms an overlay

Overlay networks are not limited to P2P networks. For instance, Resilient overlay network [7] uses overlay networks to define an application based routing scheme over the Internet. Overlay networks are also used to model the path of a multicast network. Traditionally, graph based modeling schemes were utilized to build the path of a multicast network however, they failed to completely model the random distribution of nodes resulting in delayed lookups and unguaranteed search results. This is mainly because the graph does not holds a correlation with network topology. Hence, different mechanisms like maximum number of hops a query travel or maximum time to live for a packet are employed to control the network traffic. However, network is random and hence even though a node with the required data is present it is not guaranteed that the graph will cover the node. Overlay networks score over the graph based scheme mainly because it provides guaranteed delivery of data and that too in the fewest hops. For example Chord [1] has a circular overlay of nodes and uses a maximum of $O(logN)$ routing messages where, N is the number of nodes.

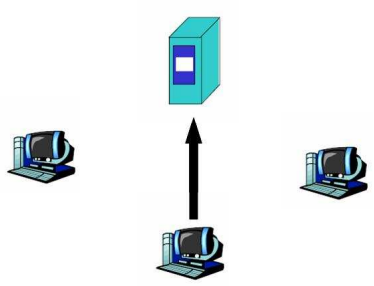
2.3 Types of Peer-to-Peer Networks

This section provides an overview of different types of P2P networks and their search mechanism. These examples will further help in understanding the actual problem statement we have explored in this thesis.

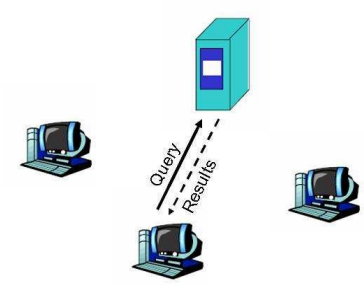
2.3.1 A Hybrid Peer-to-Peer Newtork - Napster

A hybrid P2P network is a combination of a pure P2P network and a client-server model. This means that the participating hosts in the network plays the role of both a server as well as a client. But unlike a P2P network, there is also presence of a central entity (server). This section discusses one such P2P system - Napster [8]. Napster is chosen mainly because it was the first commercially available P2P application. Napster is a perfect hybrid model that uses client-server system to provide file sharing services.

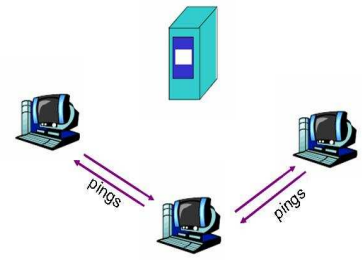
Napster was originally design to share music files over the Internet. Napster has two major components 1) a centralized directory server and 2) an application level point-to-point communication. A host joins the network by connecting to the central server and uploading list of all the shared files. The joining host also provides the keywords for searching its shared files to the server. To search a file, a querying host sends a request to the central server as a keyword. The central server scans its list of files and replies to the host with the list of IPs of hosts sharing the requested files. The requesting host then initiates a direct P2P connection with the hosts which has the best transfer rate. This process is depicted in Figure 2.3.



(a) A peer requesting for information from a napster central server



(b) Napster provides a list of peers who have the requested information



(c) Peers transfers data by establishing a direct connection with other peers

Figure 2.3: Napster - a hybrid peer-to-peer model

2.3.2 An Unstructured Peer-to-Peer Network - Gnutella

As discussed in section 2.1, Gnutella [9] is one of the most popular P2P network originally modeled on unstructured P2P networks. A commercial version of Gnutella is available as Limewire [10]. Gnutella provides a very good example of decentralized searching mechanism. Since it is built over a pure P2P network, there are no central servers in the sytem. Every host in Gnutella are describe as "servent", derived from the words server and client, and share files with the network. Other peers can then access these shared files. A search in a Gnutella network is similar to the hypothetical society described in Chapter 1. Hence, if a peer A is searching for a file X, it will broadcast a query request to all his neighbors on the network. The query then propagates recursively across the network until a host is found who shares X. Figure 2.4 depicts an usual query broadcast scheme in a Gnutella network.

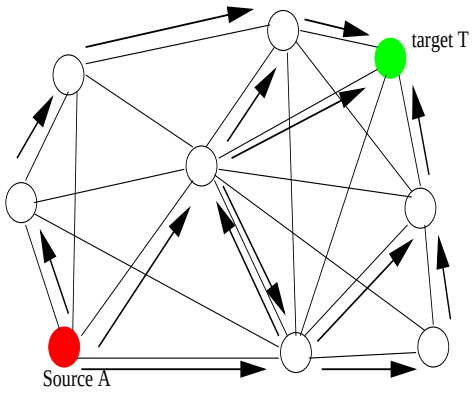


Figure 2.4: A usual broadcasting of messages in Gnutella from a source to a target peer

2.3.3 A Structured Peer-to-Peer Network - Chord

Finally, there is one more class of P2P networks called Structured P2P networks. As discussed in section 2.2, these P2P networks build overlay networks on top of P2P

networks. Chord is designed with this intention. It provides a common abstraction for all the entities participating in the network by generating an identifier (key) for all the nodes and data shared across the network. Using the identifiers a circular overlay like ring is created. Every node and data maps to the circular overlay hence, the notion of a node responsible for maintaining its shared data is replaced by *every data to be shared has a location on the network.*

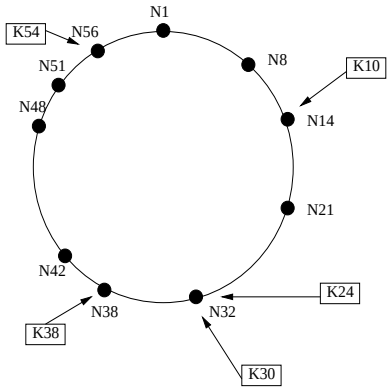


Figure 2.5: A Chord's identifier circle with ten nodes storing five keys.

In Chord, a lookup for data is performed as a search for a key. A key is mapped to a node whose identifier is equal to or closely succeeding the key. Since, overlay is a circular ring, a node starts its search by looking its immediate successor and predecessor who in turns look for their successors. In this way a query is propagated across the ring until the right node with the key is found. To overcome the overhead of linear traversals, pointers to node at certain distances (also known as finger table) is maintained by every node on the identifier circle. This is depicted in Figure 2.5.

2.4 Other Related Work

Several related work have been proposed to improve the efficiency of the search in P2P systems. We have found three locality-based search schemes closely related

to our work. The major difference is the motivation in the locality definition. In Foreseer [11], a locality is defined as two orthogonal overlays - neighbor overlays and friend overlays. The motivation is to search the previously visited node first before searching the neighbor nodes. However, locality is defined geographically (neighbor overlays) and temporally (friend overlays) unlike our scheme which is a semantic categorization. Our scheme provides a hierarchical framework. For instance, L-chord provides a hierarchical interface for locality by creating a two level Chord. Also in Foreseer locality does not utilize the semantical significance of data.

In Barbosa et al. [12] , nodes are clustered based on an individual node's interest. A node joins its cluster by matching a subset of its shared files. This puts a tight restriction on a node to be a part of the locality. In our case nodes are not hard bound to a locality. Instead, nodes share their data by distributing them over the locality overlays. Thus, the constraint is placed on the data and not on the nodes. Moreover, the nature of a node's locality is not defined by matching the content of files it shares, rather on the nature of the queries it generates. This satisfies a node's changing requirements for different types of queries as well as its ability to share its contents with the multiple localities. The search domain of a node is thus, considerably increased. The peers are not explicitly transferring data to become part of the locality. Instead, peers join the network in usual way and data are placed inside the locality which shares its semantics.

KaZaA [13] also models the system as locality by introducing superpeers. In KaZaA superpeers change dynamically like L-Chord but it is determined on the basis of its popularity. Hence, a peer in KaZaA demotes from a superpeer to a normal peer depending on the popularity of data it shares. L-Chord is designed with a different philosophy. L-Chord locality representatives are similar to superpeers but their role is to route the query to the target domain.

Chapter 3

Problem Statement and Identification

The primary motivation behind this work is to improve the search performance of a P2P network. Our goal is to realize a P2P network model that exploits the semantics of the search queries. In general, a query search over a network domain, which shares the same meaning with the query, has higher search success probability compared to searching in a random domain. Thus, by targeting the search to a meaningful domain, a better search scheme can be realized. The problem identified in this section addresses the inability of different schemes to exploit the semantics leading to compromised performance.

3.1 Uninformed Search

The different P2P search models described in Chapter 2 are analogous to the hypothetical society described in Chapter 1. For instance, in Gnutella (see 2.3.2) if each node is describe as a person in our hypothetical society, then the search for a data

is analogous to a person looking for a piece of information. Imagine a person A, looking for an information X. To search for X (see Figure 3.1), A will ask all the people in his vicinity for X (broadcasting in Gnutella). A person B, in the vicinity of A who does not hold X, on receiving request will also search for X in the same way (broadcasting again). This process continues until a person D, who holds X, is found. The location of D is then tracebacked to A through multiple chains of people. This shows, that the path to the target peer is undefined and search has no clue of its path.

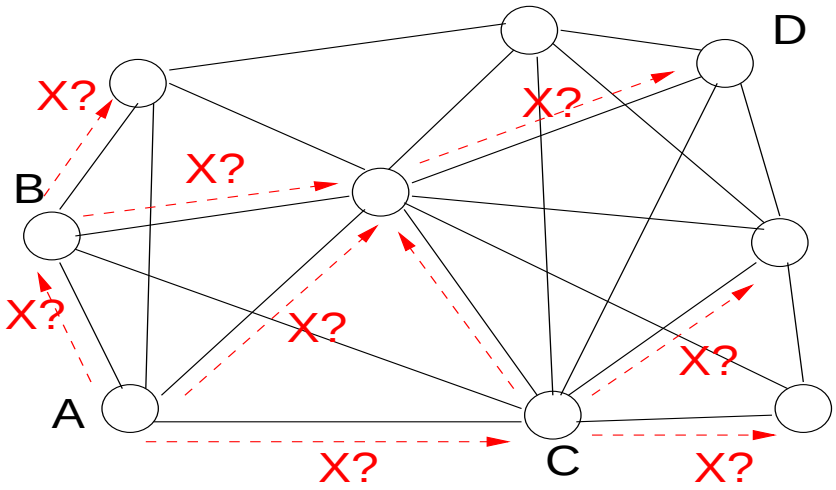


Figure 3.1: Depicts a situation of uninformed search.

3.2 Unstructured Nature

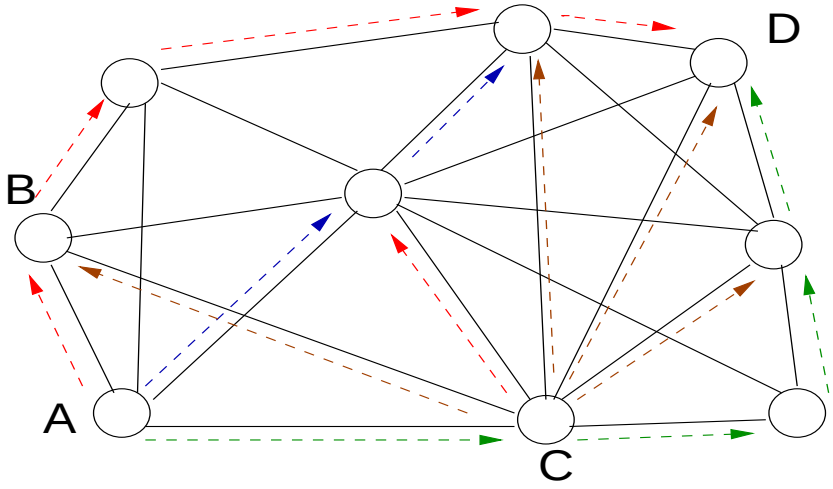
Consider existence of another person C in the vicinity of both A and B. C must have received requests for X when both A and B initiated their search. What does this imply? C receive redundant requests. Finally, assume A has X from D far away in the society. If B also wants to search for X, he will initiate a search similar to A. B will then get X from A (since A is in vicinity of B) but at the same time

B can not avoid C from looking for X or any other people in the vicinity of B (see Figure 3.2(a)). This means a person has no control over the path of the search and the search itself is blind and undirected. Further, lets assume A accidentally destroyed X and wants to re-initiate a search for X. Further, also assume no body in A's vicinity has X. What should A do now - reinitiate the same redundant process again? How can A avoid this process?

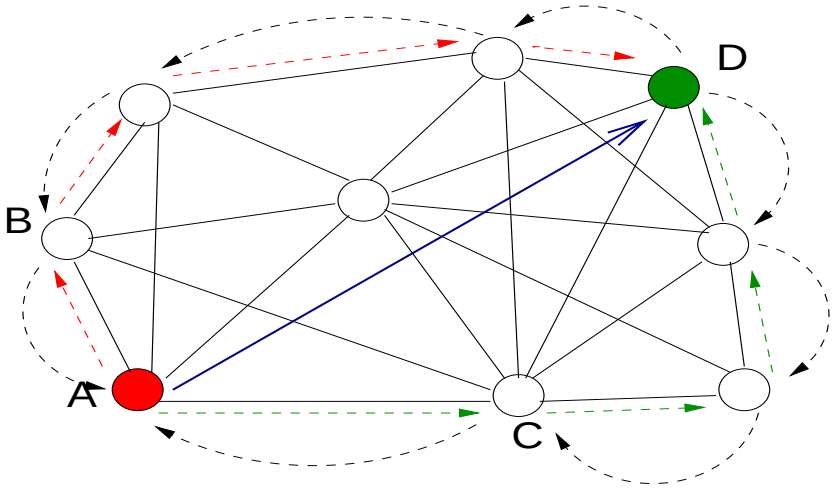
In the above discussion when A received X from D for the first time, he could also ask D for his "business card". This is similar to a scenario where a person visits different pizza shops and collects their business cards. So next time when he needs pizza, he can directly call or go to the pizza shop using their business cards. Hence, A's subsequent searches for X then merely reduced to checking with D first before broadcasting. The search technique proposed by Sripanidkulchai et al. [14] extends this philosophy and is depicted in Figure 3.2(b). Each node maintains shortcut to the peers that have previously satisfied any query. Hence, an extra piece of information "business card" greatly reduces unnecessary consumption of resources due to random searching.

3.3 Unpredictable Join and Departure Behaviors

The search philosophy described above is iterative because of the unstructured nature of P2P systems. A node gathers business cards (or shortcut indices [14]) during the course of its search and save these cards for future references. Lets assume A has business card of D for information X. If suppose, D decides to leave then A has no way of knowing that D has departed from the society. Hence, next time when A searches for X, his query to D will fail (Figure 3.3). This is one of the



(a) Multiple paths because of unstructured network. Some nodes receive redundant requests.



(b) A keeps buisness card for D for fast retreival of X in sebsequent searches.

Figure 3.2: Consequences of Unstructured Nature

common problems in a P2P system. The probability of a node departing the system is unpredictable. So even though A has business cards of different people in the society, if all of them leave, A's problem remains unsolved. Added to the problem there is no control on the frequency of peers departing and joining the network. The overhead of maintaining shortcuts will also become a burden for nodes since, every time a node joins or departs the system other nodes have to dynamically update their shortcuts as well. Moreover, the shortcuts have no significance with data.

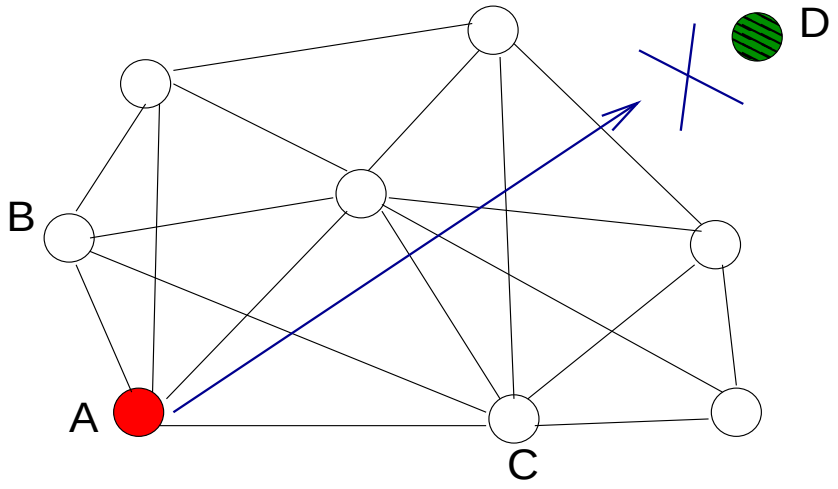


Figure 3.3: Failed search because D departed without A's knowledge.

3.4 Problem Summary

From the above discussion we have identified some of the potential problems related with the searching in a P2P system. These problems are summarized below. In general a search in a P2P system suffers from:

1. random distribution of the data leading to redundant and uninformed traversing across the network.

2. unpredictable join and departure behavior of the nodes.

3. difficulty in achieving physically close overlays of nodes sharing common data.

3.5 Proposed Solution

To begin with, let us incorporate solutions to all the different problems that were disccussed in the last section of our hypothetical society. Hence, we define in the new society, every person has business cards to look for people who have served him for any information, every person knows about their position in the society so that others can easily reach them. The new defined society is now analogous to a real world society. To define the proposed solution let us try to understand an application problem on the top of this society with the proposed scheme.

John has recently moved to Rochester and has no acquaintances. He recently had a toothache so he is searching for a dentist. How can John find a good dentist in Rochester? If through yellow pages or any other source of information (like friends, colleagues, etc), John is made aware of different "consultants" in Rochester who keep information of all the dentists, he can save himself from the burden of visiting and keeping business cards of different dentists. Moreover, John will have access to an information center which provides information about dentists.

This approach has two important characteristics: 1) When different dentists start their clinic, they have to make themselves aware of their existence to the consultants. 2) It is not necessary for dentists to inform only one consultant. There can be many consultants based on a region and they all can have information of same or different dentists.

If we expand the role of consultants by not limiting them to be a distinguished "Dentist Consultants", then they can act as a known point for information access. This means they can also have information of "Pediatrics", "Florists", etc. Hence,

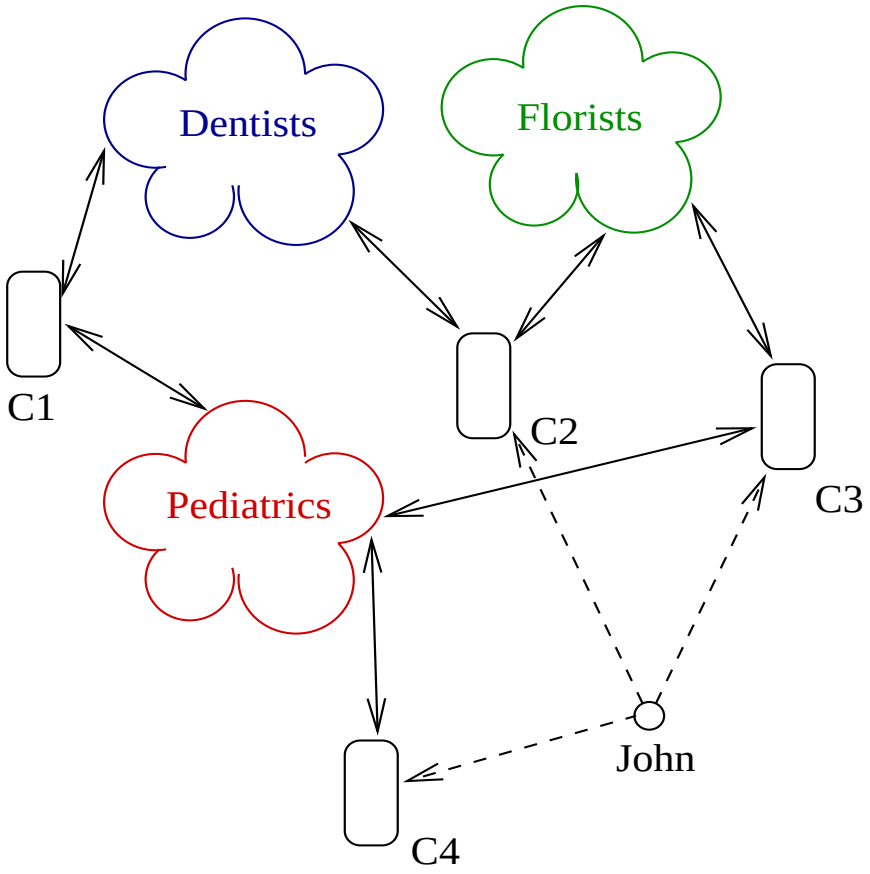


Figure 3.4: $C1$, $C2$, $C3$ and $C4$ represents different consultants for different localities shown.

the society can then be viewed as different "localities" of dentists, pediatrics, etc centered around different representing consultants. If we know the semantic meaning of the information being searched (like dentists), it is possible to realize a similar network configuration with different localities. For instance, we can have locality for dentists, florists, etc. This is depicted in Figure 3.4. To adapt this theory, we propose a framework that exploits the semantics of data to model a P2P network as classified "localities" of information.

3.6 Summary

This chapter provides the motivation for this thesis. Sections 3.1, 3.2 and 3.3 provides discussion on different types of drawbacks with unstructured P2P systems. Section 3.5 provides a clear picture of locality scheme by giving a real world example. Thus, it has been demonstrated that by categorizing the search domains based on semantics, search can be targeted to domains with high success rate probablility with no redundancy. Finally, this chapter introduces the concepts required for the description of the proposed locality based scheme - L-Chord discussed in the next chapter.

Chapter 4

L-Chord - A locality of Chord

4.1 What is a Locality?

In the last chapter, we proposed a P2P system as classified localities of information. A locality is a logical semantic based overlay of nodes sharing similar types of data. This is depicted in Figure 4.1. The goal of our work is to provide a mechanism for creating such localities similar to the society situation described in the problem solution (section 3.5). L-Chord is designed on this philosophy. L-Chord ensures data are placed inside locality of their semantic significance. This is done by determining the properties of the data provided by the end user. In case, when user does not provide any properties, L-Chord tries to identify the locality on its own.

Figure 4.2 describes the organization of a L-Chord based P2P network layers. The bottom layer defines the underlying network protocol for communication. The middle layer is defined as the logical layer. This is the layer where Chord resides. The top most layer implements our proposed model - L-Chord.

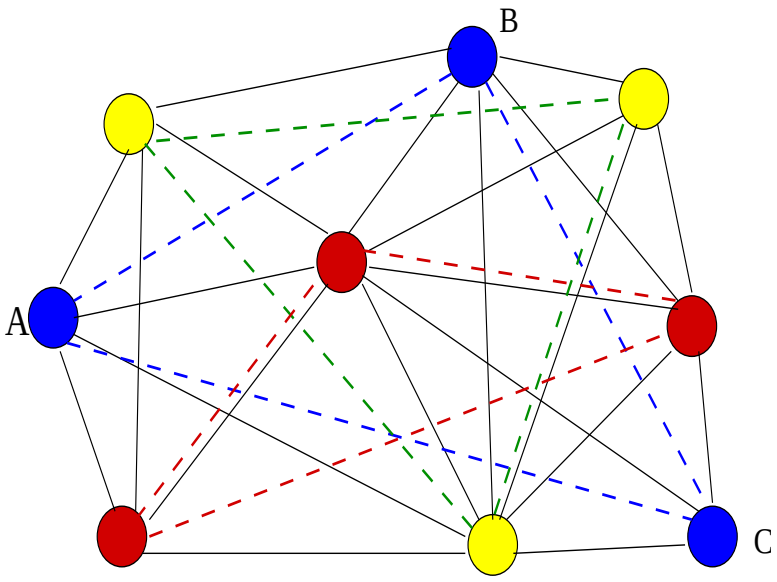


Figure 4.1: An overlay of localities on top of a P2P network shown in colors.

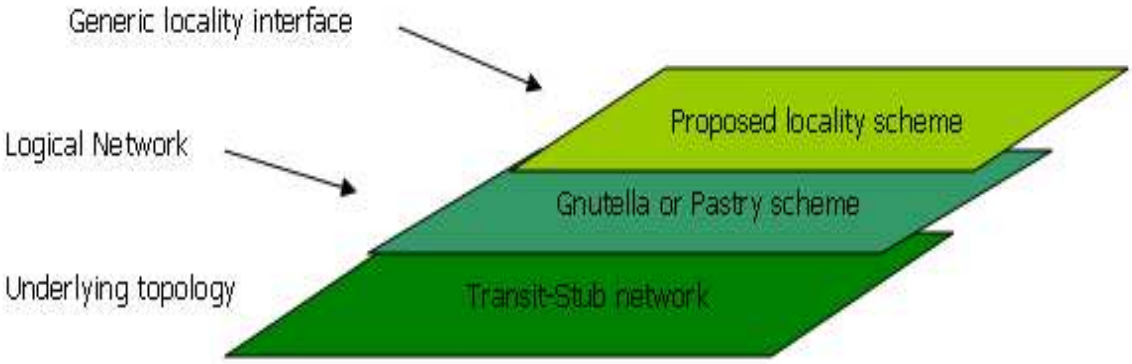
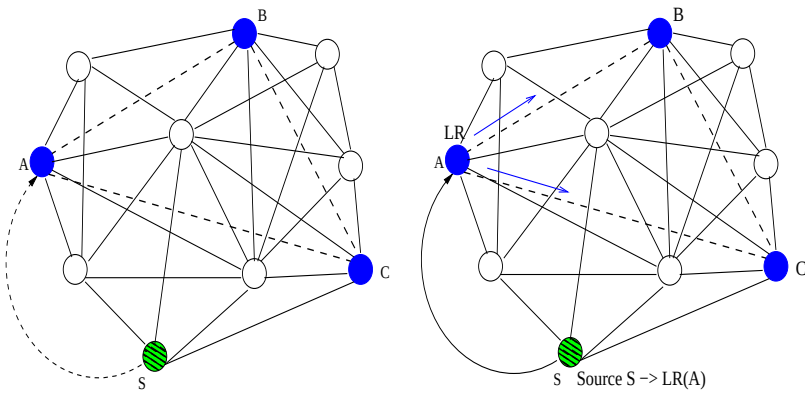


Figure 4.2: A Layered view of the L-Chord based P2P Model

4.1.1 Locality Representative

A locality representative (L-Rep) is a special peer defined in the system. Any peer can play the role of a L-Rep (consultants see 3.5). A L-Rep holds the information for multiple localities across the network. Whenever a search is performed, a L-Rep is determined first. During the network boot up, no peer has information of any locality. In this case when a peer queries for the data, the underlying P2P search mechanism (layer 2 in Figure 4.2) determines the location of the peer containing the data. The peer thus found, is designated as L-Rep for the query semantics. The query initiating node then holds a reference to the newly discovered peer. This is depicted in Figure 4.3. Every subsequent search for the data is then made to pass through the associated L-Rep unless there is no representative defined for that query semantics.



(a) Source node S finds a L-Rep using the underlying search mechanism

(b) Source S has now a pointer to L-Rep (A)

Figure 4.3: L-Rep discovery scheme is shown

4.1.2 Locality Formation

This section provides the basic methodology for creation of localities. As discussed in the last section a L-Rep, acts like a service provider for a peer seeking a data inside a locality. We now propose that the localities should always be defined in respect to at least one L-Rep. This helps a L-Rep to target the search specific to its locality. Figure 4.3(a) depicts a locality overlay comprising of nodes A, B and C. Note a peer can act as L-Rep for multiple localities. To enforce the locality overlay, the data shared by a peer is intentionally distributed across different localities. This is performed by first determining the L-Rep for the data based on its semantics. The data is then routed to L-Rep in single hop as a key. Finally, L-Rep maps the data to a node inside the locality. This process is shown in Figure 4.3. Node S stores a data as key (X) by first routing X to A who maps the data to any of the nodes B or C since they all are part of the same locality. Note it is also possible for L-Rep to store data. This is because a L-Rep is a normal peer like any other peer in the network and is also part of the locality it represents.

4.2 What is L-Chord?

L-Chord is a two-level Chord ring. A L-Chord is viewed as "Chord of Chords". The two-level ring is built on top of the Chord's overlay. At the first level L-Chord contains different L-Rep peers. The second level defines a locality for every L-Rep peers. The two level scheme helps a query to reach the target peer utilizing its query semantics. Figure 4.4(a) provides an abstract view of L-Chord when viewed as a group of localities on top of the Chord ring. Figure 4.4(b) gives an alternate view of L-Chord as different small Chords forming a big Chord ring. A locality as shown in Figure 4.4(b), is an overlay of different peers containing similar data types. Note

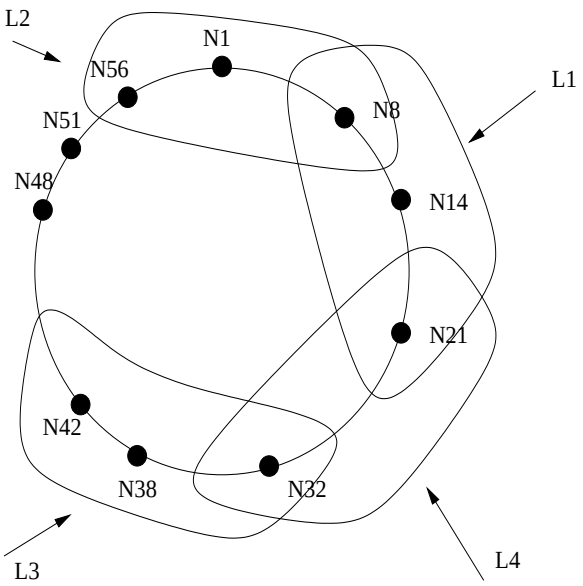
unless otherwise explicitly stated, the term *similar data types* is defined as the data belonging to the same locality. For instance, if A and B are two different data with same locality L then, A and B are considered as the same data type. In other words if $L = Audio$, then A and B are both "Audio" files belonging to *Audio* locality.

4.2.1 Semantic Evaluation

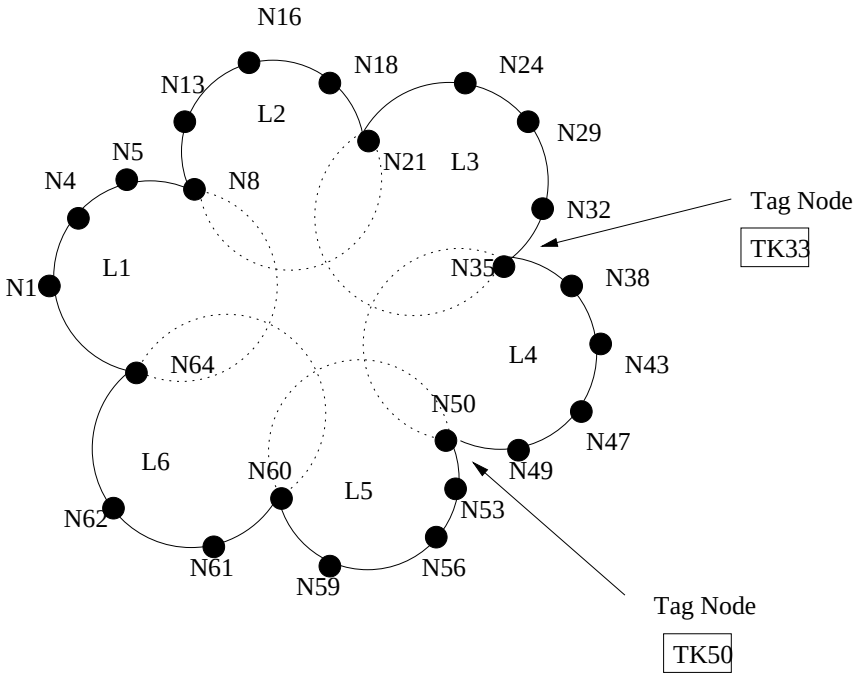
Semantics of a data is defined as the set of properties that can identify its uniqueness under the classification of localities. For instance, a file X can be a video file (with extensions .mpg, .mov, etc) or it can be a audio file (with extensions .mp3, .wav, etc). Only by examining the file extension, X can be identified as an audio or a video file. However, there can be two different files with the same extension but different localities. For example, both "X.wav" and "Y.wav" has the same extension but they can be either a audio or a video file. This is because ".wav" extension is used for both audio and video files. In such a case, file type is determined by reading its header bytes. For instance, WAV file format is composed of different chunks for describing waveform, sample rate, etc which are used to determine its properties, a java class file starts with a $0xCAFEBABE$ on big-endian systems. The examples given demonstrates a technique to determine semantics of a file. Other techniques can also be employed such as metadata(or XML tags).

4.2.2 Locality Identification in L-Chord

L-Chord ensures data are stored in the system based on their semantics. Hence, a locality is determined for every data before they are stored in the system. In general before a search is performed, a user is provided with pre-defined locality options like Music, Video, etc. Similarly, when a user stores a file, user provides the file characteristics and a locality is determined. In case a user does not provides



(a) A L-Chord can be viewed as overlays of locality over Chord



(b) Another view of L-Chord viewed as Chord of Chord

Figure 4.4: A Chord with localities. Every locality groups similar type of data. Nodes can be a member of multiple localities.

any file characteristics, locality is determined by doing the semantic evaluations described in the last section. Every locality is uniquely identified by its *locality-tag*. A locality-tag is a string comprising of alpha-numeric characters use to determine the representative for a locality. Using locality-tag search is routed to the representative node in one hop.

4.3 Adaption with Chord

L-Chord uses Chord's paradigm (DHT [15] specifics) for mapping data to the network. Data is mapped to the localities as keys. Similarly, a key for the data is generated when a search is performed.

4.3.1 L-Rep Peer Identification

L-Chord defines a locality-tag as the representative of a locality. If locality-tags are treated as Chord keys then associated L-Reps can be determined on the Chord's identifier ring. For every locality-tag, consistent hashing is used to generate a "tag-key". A tag-key is mapped to a node on the Chord's ring whose identifier is equal to or greater than the tag-key. In this way the integrity of Chord's key mapping mechanism is preserved. The node found for the tag-key is called as "tag-node". Every peer maintains list of all tag-keys that represents different tag-nodes(L-Rep) on the network. This is depicted in Figure 4.4(b). As shown, the first level of L-Chord is created as an overlay of tag-nodes.

4.3.2 Locality Overlay Creation

A tag-node hence, is a L-Rep peer around which a locality(sub-chord) is created(small circles in Figure 4.4(b)). The data is then mapped to the sub-chord ring using the

Chord's mapping scheme. The mapping of data to sub-chord ring is explained below:

1. For every data or a file, a key is generated using the Chord's API.
2. Instead of nodes directly mapping the keys to the Chord ring, tag-node maps the key over the sub-chord ring.
3. The above step ensures a key is always mapped to a node who is in the sub-chord identifier space.

Thus, we define a sub-chord as a small Chord ring where files are inserted, retrieved, and routed as the original Chord. All the nodes in the sub-chord contains similar types of data(as required by a locality). A sub-chord has constant length of 2^{m_1} where $m_1 < m$. Within the sub-chord, only local control traffic is exchanged to maintain the sub-chord insert, lookup, and routing mechanisms. Similarly, the control traffic for the first level Chord travels only around a peer originating the query and the tag-node. This property increases scalability, since a node in the sub-chord maintains data specific to the locality(local Chord ring). The algorithm(Algorithm 1) for tag-node lookup is then merely defining the tag-key and looking up for the tag-node using the Chord's API.

Algorithm 1 Algorithm for tag-key lookup $n.locality\_lookup(tag - key)$

return $n.find\_successor(tag - key)$

4.4 Routing in L-Chord

As described, L-Chord is a two-level logical ring structure. The messages in the L-Chord space are routed across the first level to the second level where the target peer resides. Every message in a sub-chord is routed towards the node with the closest

ID in $O(\log n)$ steps using the finger table where, n is the number of nodes($= 2^{m_1}$) in the sub-chord. It is important to note that L-Chord maintains the integrity of the Chord. No changes are made to the placement mechanism of the nodes on the Chord's ring. The two levels are only logical adaption of locality scheme over Chord.

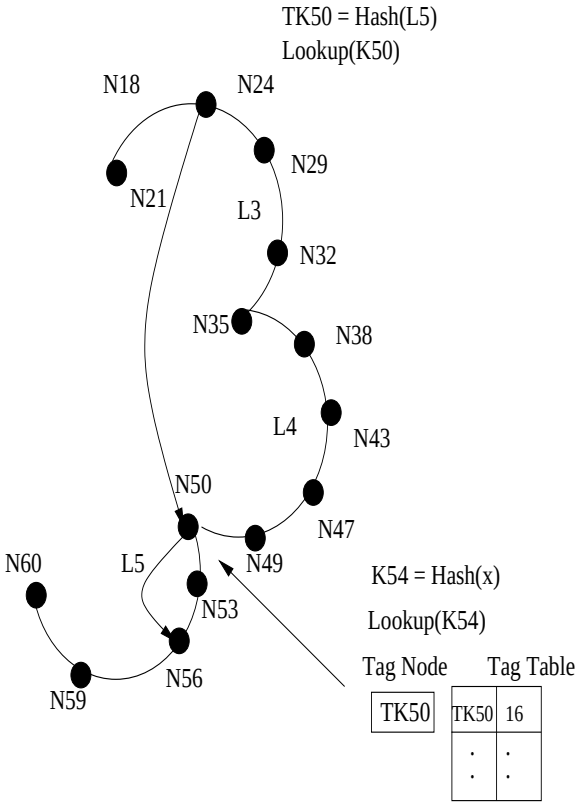


Figure 4.5: Lookup operation in L-Chord. Node 24 finds the tag-node for locality L5 (tag-key 50) at Node 50. The tag-node 50 routes the query to sub-chord L5 starting at Node 56. The Tag-Table for tag-node 50 is also shown

Figure 4.5 demonstrates a normal routing over the L-Chord. Node 24 determines its locality-tag $L5$, and routes the message to the tag-node 50. Node 50 then determines node 56 as the target peer in the locality for $L5$. In this way a message is routed from the node 24 in the first level to a sub-chord defined by locality $L5$.

If we compare the routing in L-Chord with the original Chord scheme, the first level routes only one message to the tag-node. Tag-node forwards this message to its

sub-chord using the original Chord protocol in $O(\log n)$ messages, where $n < N$ and N is the size of the original Chord ring. Thus, total messages routed are $1 + O(log\, n)$ which is an improvement over the original Chord routing messages($O(log\ N)$).

4.5 Data Storage Mechanism in L-Chord

The effectiveness of L-Chord is realized as the data distribution over the sub-chord ring. The data storage procedure closely follows L-Chord routing mechanism (described in 4.4). A peer shares its data with the network by first determining the tag-node for the data and then storing inside the sub-chord ring. A data file is consistent hashed modulo 2^{m_1} and mapped to the sub-chord (length $= 2^{m_1}$) space at a node whose identifier closely succeeds or equal to the distance $d = tag - key + key(mod 2^{m_1})$, where $tag - key$ is the identifier of locality on the first level L-Chord space and key is the data mapped to the sub-chord. In this way data are grouped together as they are stored over the sub-chord space to form a locality. It is important to note that the distance d is calculated in distance with the tag-key and not the tag-node. A consequence of calculating d as a distance from tag-node is explained in the section 4.7. Also, it is possible for a node to store same data keys for different localities because of the way sub-chord mapping is defined. However, they can be easily distinguished by their tag-keys. Algorithm 2 describes the algorithm for the store process.

Figure 4.6 depicts the data store procedure in L-Chord. File X can be an audio or a video file based on its properties(for example file extension such as .mp3, .wav, etc). On examining X's properties, it can be further classified according to its genre(rock, pop, metal, etc). Let us assume, we define a locality "Rock Music" with locality-tag "RM" and X is an audio file of "rock" genre. To store X, RM

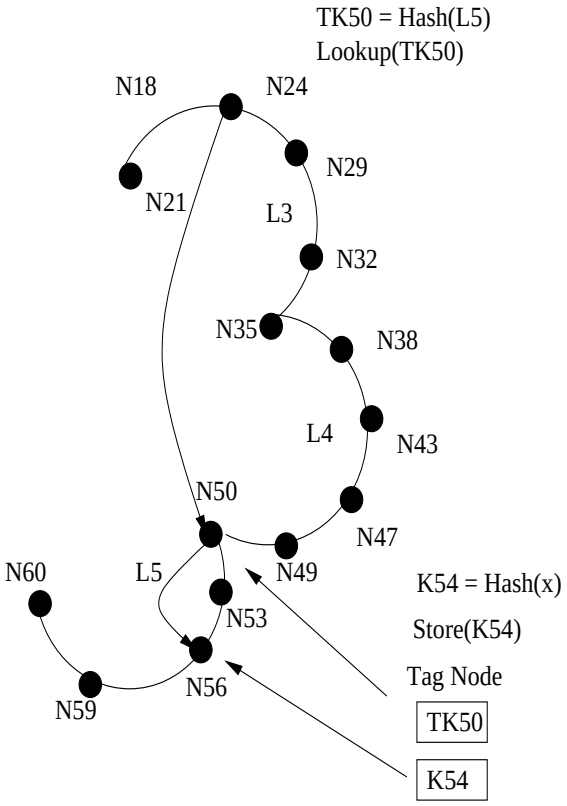


Figure 4.6: Node 24 has data "X" to share. The tag-node for "X" is determined using L-Chord lookup protocol and data "X" is consistent hash (Key 54) and stored at Node inside the L-Chord.

is first hashed to obtain a tag-key (50) as shown in Figure 4.6. In the Chord ring, node 50 has identifier equal to the tag-key so it becomes the tag-node for X. If we assume the length of the locality "Rock Music" as $16(2^4)$ and the key for the X as $4(modulo16)$ then X will be stored at node 56. Since, X's key $54((50+4\ modulo16)$) has node 56 as the closest successor inside the sub-chord ring.

Algorithm 2 Algorithm for storing a key in sub-chord space $n.store(tag-key, x)$

```
n' = tagtable_lookup(tag − key)
if n' not found then
   n' = n.locality_lookup(tag − key)
end if
n'.store(key(x))
```

4.6 Lookup Mechanism in L-Chord

The Chord protocol requires a lookup to be performed over a key already present on the newtork. This is because Chord destroys the semantic significance of data by generating keys. L-Chord is a framework designed with the intention of imposing a locality model on top of the Chord ring. Our goal of is to evaluate the success rate of a query targeted to an overlay which shares same semantical meaning with the query. Hence, to lookup a query in L-Chord, the messages are routed with the Chord's search constraints. Figure 4.5 demonstrates a lookup operation similar to routing in L-Chord. This figure also shows a "tag-table". A **Tag-Table** is defined as a table of tag-keys with their corresponding tag-nodes. This table is updated every time the mapping of the tag-key is changed. Figure 4.5 demonstrates a lookup similar to the data storage procedure. But instead of storing, the lookup process only determines whether the query is satisfied at the target peer inside the locality.

For simplicity, assume there is a peer A looking for a data X. To lookup X, locality-tag for X is determined and tag-key is calculated. A then perform a lookup in its tag-table to check if there is a known locality present. The query for X is then routed to the tag-node who in turn routes the query to the sub-chord using original Chord protocol. In case a node does not have information for a tag-node, a lookup(using the original Chord lookup protocol) for the tag-key is performed. The node thus, found is promoted as the tag-node for the semantics defined by X. Finally, A updates its tag-table and lookup is routed inside the sub-chord by the tag-node. Algorithm 3 provides the pseudo code for this process.

Algorithm 3 Algorithm for key lookup $n.exlookup(x)$

```
tag − key = getTagKey(x)
n' = tagtable_lookup(tag − key)
if n' not found then
  n' = n.locality_lookup(tag − key)
  update_tagtable
end if
return n'.find_successor(key(x))
```

4.7 Join and Departure Mechanism in L-Chord

L-Chord framework is designed to exploit the search scheme of underlying Chord. Hence, L-Chord does not impose any changes to the join and departure mechanism of Chord. This mechanism is Chord dependent. The only extra information L-Chord adds to the Chord is the notion of tag-nodes. Since, tag-keys directly map to tag-nodes, if a tag-node departs from the system, the tag-table for every peer has to be updated. For instance, assume node 10 as the tag-node for the tag-key 8. If a new node 9 joins the system then, the tag-table needs to be updated since tag-key will map to the new node. Similarly, if the tag-node 10 fails then the tag-key

has to be moved to tag-node 9 (or node 11 if node 9 is not present). To keep the framework simple a tag-table is only updated when the tag-node itself departs from the system. Hence, when a jump to tag-node fails, a new tag-node is determined and tag-table is updated for the corresponding peers. A node only knows about the new tag-node when it queries for the tag-key. This situation is similar to the boot mechanism described in the section 4.1.1.

4.8 Summary

In this chapter we have provided the working of L-Chord. A locality is a close overlay of nodes containing data with similar semantics. L-Chord is a locality based framework on top of the Chord. In a large Chord system, L-Chord considerably saves bandwidth and the number of hops to reach the target peers. The system is scalable and requires little overhead of tag-table maintenance. The cost of the tag-table maintenance is less compared to the number of routing messages saved. The advantages of L-Chord is summarized below:

1. L-Chord targets the query to locality specific to its semantics. Hence, blind and uninformed search are avoided.

2. L-Chord is fault-tolerant to abnormal join and departure of the peers as multiple localities of same type exists across the system.

3. Even if a tag-node fails, the system can define a new tag-node at the cost of one underlying layer's search mechanism.

4. The number of hops required to reach a peer containing data are less.

5. Problem of redundancy is removed as the search is informed about its destination.

6. The search domain is narrowed and is more specific to the target.

Chapter 5

Results

5.1 Simulation Overview

The simulation results discussed in this section evaluates the effectiveness of L-Chord. We have used P2PSim [16] simulator - a P2P network simulator developed at PDOS Lab, M.I.T. The simulator is written in C++ and is an open source software under GNU License. The simulator provides three different implementations of Chord - Chord (with successors), ChordFinger (with successor and finger table) and ChordFingerPNS (with PNS fingers) [17]. Currently, the most stable release is the version for ChordFingerPNS. So the results in this chapter were generated on ChordFingerPNS. Note L-Chord does not have to modify any of these variants of Chord except for the routing described in Chapter 4. The three versions of Chords differs in the way routing optimization is performed. For instance, Chord nodes uses only its successors list and no other information when routing a message. ChordFingerPNS also maintains an additional PNS (Proximity Neighbor Selection) table. Both scheme applies their own mechanism but PNS performs routing based on the estimated latency. However, L-Chord does not concerns with any of these

factors as it is designed with different motivation. The simulation is performed on a Pentium Mobile 1.5 GHz Inspiron notebook with 1GB RAM running a Ubuntu (Debian Linux) 6.06 LTS.

5.2 Network Topology

The topology file used for simulation is the orignal king data set used in the Vivaldi [18] experiment. The king data set is a network comprising of 1740 nodes (available at [19]). King data set is generated using King tool developed at University of Washington, Seattle. King [18] tool calculates the estimated latency between any two Internet hosts using a third client host. ChordFingerPNS adapts the king topology well because it tries to optimize the routing by considering the latency between different hosts. Moreover, the data set is decentralized and is an ideal topology for P2P simulations. The king data set calculates the latency between any two Internet hosts using the latency between their domain name servers because in most of the cases, the domain name servers are kept near to the hosts. King topology can be described as a Edge-to-Edge graph with latency between any two nodes as the weight of an edge.

5.3 Simulation Parameters

The parameters controls the simulation. In simulation several parameters needs to be taken in to the consideration to achieve results close to live systems. Table 5.1 lists the parameters required to configure the simulator. We kept fixed values for the listed parameters during the entire simulation. Before going through the parameter description, it is important to note that simulator provides two different types of simulation. In the first type, the simulation executes only once for the given

Parameter	Value
protocol	ChordChord
base	2
nsucc	16
pnstimer	36000
basictimer	1000
succlisttimer	8000
recurs	1
initstate	1
ipkey	0
lifemean	600000
stattime	3600000
deathmean	3600000
exittime	21600000

Table 5.1: List of P2PSim parameters and their values

Parameter	Description
localsize	Locality Size
tagenable	Enable Locality based scheme

Table 5.2: List of added parameters and their description

time period and generate one time statistics. In the second type, the simulation is performed n number of times to generate samples for further analysis (like graph base). The parameters are further classified in to three different categories explained in the following subsections.

5.3.1 Protocol Specific Parameters

The parameters specific to a protocol are listed in table 5.1 and table 5.2. All the timer based parameters follows exponential mean distribution. To incorporate L-Chord some additional parameters are added shown in table 5.2.

5.3.2 Topology Specific Parameters

These parameters defines the underlying topology for the simulation. A topology file is feeded to the simulator with its name and the "failure model". The failure model simulates the link behavior between two different nodes in the simulation. For, instance Constant failure model increases the delay between two randomly chosen links. Null failure model consumes the packet destined for any source. The results presented here are evaluated on the failure models described above.

5.3.3 Event Generation

These parameters controls the generation of keys and other timing events for the simulation. It uses some of the protocol parameters for controlling simulation. The only extra parameter added to event generator is no_of_locals which defines the number of localities to simulate. The simulation performs both the lookup and store procedures simultaneously. Lookup and storage keys are randomly generated with exponential distribution. The localities are predefined and to generate tag-keys random alphanumeric strings are generated and consistent hashed.

5.4 Simulation Setup

Simulation is performed in two different ways - 1) by varying the size of a locality 2) by varying the number of localities. In the first case, locality size is varied from $100-500$ and samples were collected. The simulation ran for approximately 22 hours with a fixed base size ($base = 2$). In the second case, the number of localities are varied. Five samples were collected with number of localities ranging from $2-10$. The locality size was kept fixed to 200 ($localsize = 200$). Due to the limitations of the simulator, Chord identifier space is fixed to 2^{64} nodes. The simulation is then

performed for determining the successful lookup rates with the variation in numbers and size of the localities. The effect of localities on average path length, average latency and average stretch are also evaluated.

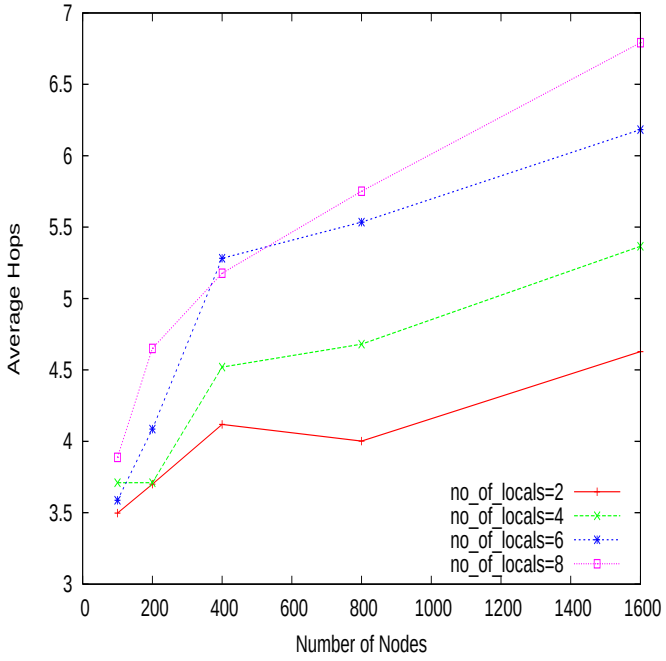
5.5 Simulation Results

The first test was performed to evaluate the dynamics of query with respect to the increase in locality size. The number of the localities were kept fixed with $base = 2$ and $no\_of\_locals = 4$. The size of locality is varied in the range of [100 - 500]. In the second test, the locality size is fixed with $localsize = 200$ and number of localities are varied in the range of [2 - 10] (increments of 2). Five samples were collected with each sample containing three different simulation result obtained by varying the stabilization timer.

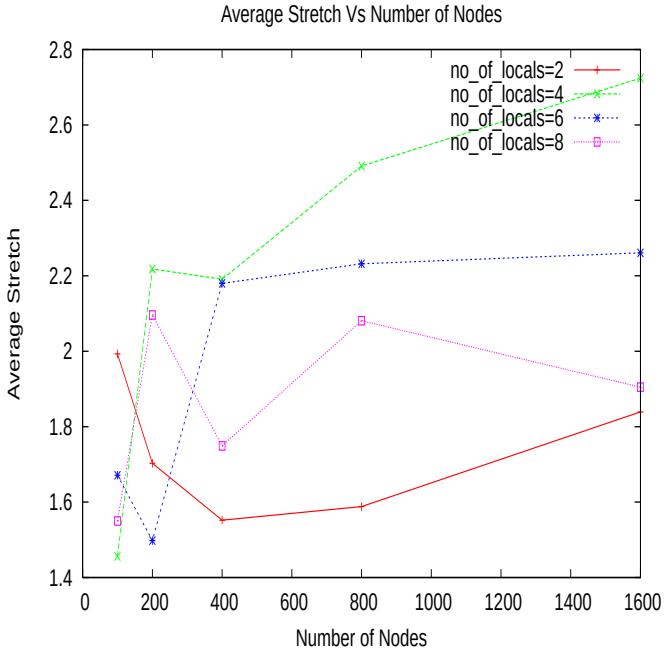
5.5.1 Path Length analysis

Figure 5.1 shows the graph for average hop. The graph also shows the variation of mean stretch with the variation in number of nodes.

Figure 5.2 shows the variation with locality size. It is interesting to observe that the plot shows heavy concentration in the lower range with some shoots. This indicates that once Chord adapts the framework and the tag-table has decent number of tag-nodes referenced, the mean hops tends to gather around the lower value. This is because the overhead of searching across the entire Chord ring tends to decrease. The tag-table ensures a single hop lookup which greatly reduces the average path length and the average stretch.



(a) Average Hops Vs Number of Nodes (N)



(b) Average Stretch Vs Number of Nodes (N)

Figure 5.1: $succs = 16$, $base = 2$, $localsize = \frac{N}{2}$

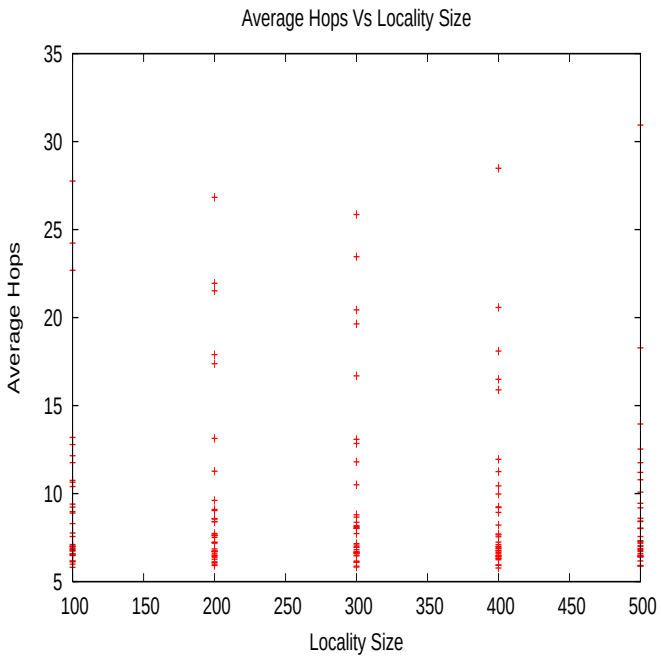


Figure 5.2: $succs = 16$, $no\_of\_locals = 4$, $base = 2$

5.5.2 Success Rate

Figure 5.3 shows the variation in success rate with variation in number of nodes. For any variation with number of localities, the success rate is close to 100%. Although there is a variation with low number of nodes but L-CHord achieves a success rate of over 99% in all the different cases. We have observed during the simulation, for low number of lookups, the success rate is low. This is because number of lookups are consumed by the large failure in nodes. In such a situation the simulation spends more time in recovering the system resulting in increase in number of false lookups because of the unstabilized finger table. For instance, a query might end up looping around the ring until all the nodes run their stabilization protocol and fix their finger table. Specific to this simulation with the random seed used for the lookup we have observed, for number of lookups above 10000 the number of node failures are less and hence, we can see higher success rate. Figure 5.4 shows high concentration of

point closer to 100% success rate ensuring L-Chord preserves the Chord integrity.

From now onward we will refer to the total lookups as the total number of successful lookups for the right node which may have the data. It is possible that even though a node lookup is successful, the node may not have the correct data. We have design the system to achieve a better search performance. Although, data storage analysis is performed in this chapter but no constraint are made on the datakey generation during the simulation. Chord ensures that given a datakey already mapped on the network, the lookup gaurantees successful lookup of the nodes containing the data. In our simulation, the datakeys are randomly generated and hence a probabilistic measurement on successful data lookup can not be guaranteed. To achieve a more probabilistic result, datakeys can be mapped to the nodes first and then lookup can be performed for only the datakeys which are already present. However, this will voilate the orignal motivation of this thesis to evaluate the system as a P2P search model.

5.6 Routing analysis

We further insist for L-Chord to have optimized successful lookup rates, the size and number of localities must be defined carefully. We propose that the size of the locality times the number of localities should fall within the permissible range of the number of nodes present in the system. In the simulation results we have ensured that this relation holds for all the experiments. To verify the proposed relation, let us assume we have a Chord ring of size 50 with $no\_of\_locals = 4$ and $localsize = 100$. This means that we have layout our network as four different localities each mapping to 400 nodes. Since, the number of nodes present is less, the lookup for tagnode will add extra overhead in a query lookup. In such a situation

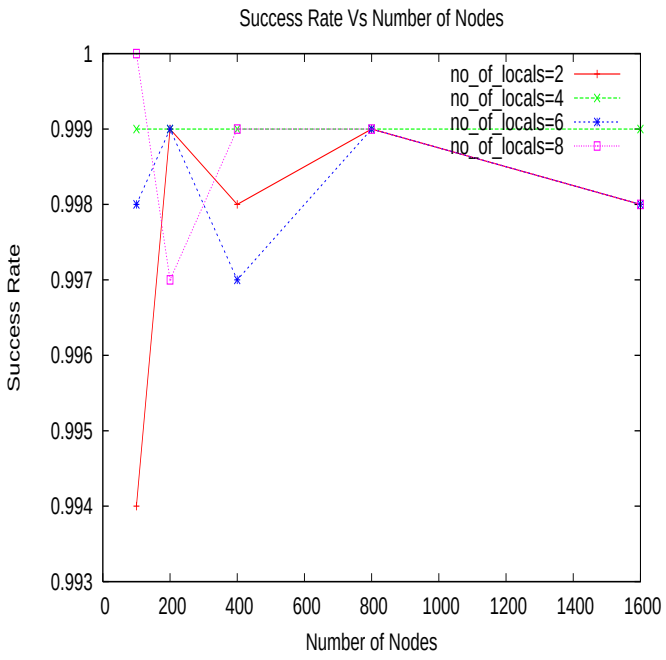


Figure 5.3: Plot between Success rate Vs Nunber of Nodes (N). $base = 2, succs = 16, no\_of\_locals = \frac{N}{2}$

original Chord lookup will perform better than the L-Chord. Ideally, we want to realize the following equation

$$1 + log(n) \leq log(N)$$

where, n is the number of nodes in a locality and N is the number of nodes in the Chord ring and $n \leq N$. The following equation is derived from the fact that Chord take $log(N)$ messages to perform a search. Our model forces this search mechanism only within a sub-chord and one additional hop for tagnode lookup from the tagtable. From the above equation we have

$$\frac{n}{N} \leq \frac{1}{2}$$

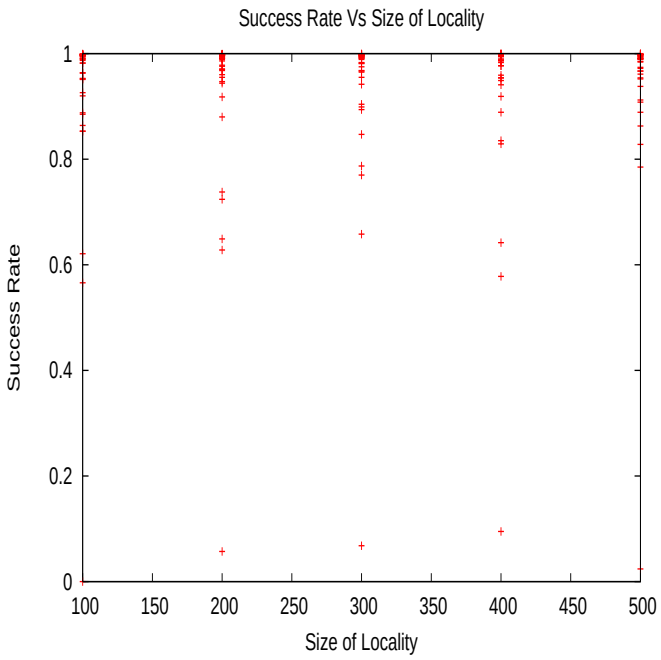


Figure 5.4: Plot between Success rate Vs Size of Localities. $base = 2, succs = 16, no\_of\_locals = 4$

. Hence, if we have l number of localities, the relation between the number and the size of locality can be derived as

$$l * localitysize \leq N$$

In the worst case a lookup for tagnode will fail which will incur an additional $log(N)$ overhead. Hence, on an average lookups will propagate in

$$log(n) + log(N)$$

routes. For $N >> n$, number of routing messages will approximate to $log(N)$ which is not greater than Chord's routing messages. If we have $n \leq \frac{N}{2}$, then we have $2 * log(N)$ messages which is same as $O(log(N))$.

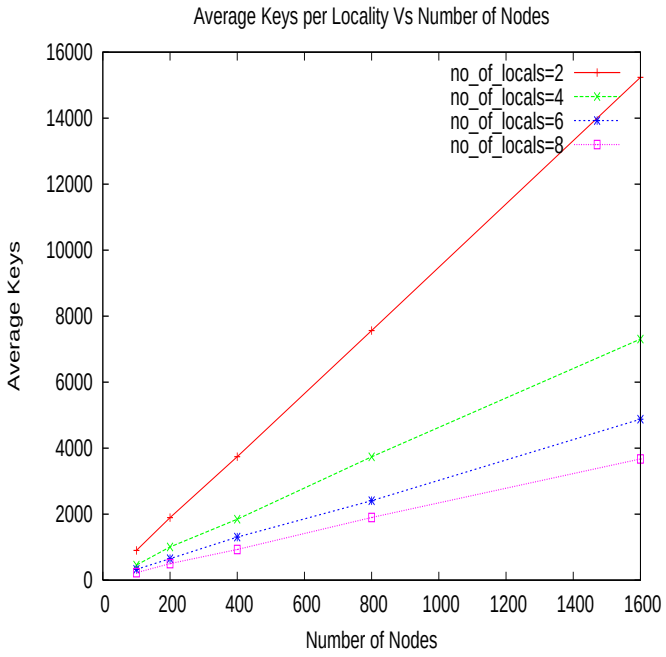
5.7 Data Storage

Figure 5.5 shows the graph for distribution of datakeys over every locality and per nodes for varying number of nodes in the topology. Figure 5.6 shows similar comparison with varying locality size for a network of 1740 nodes.

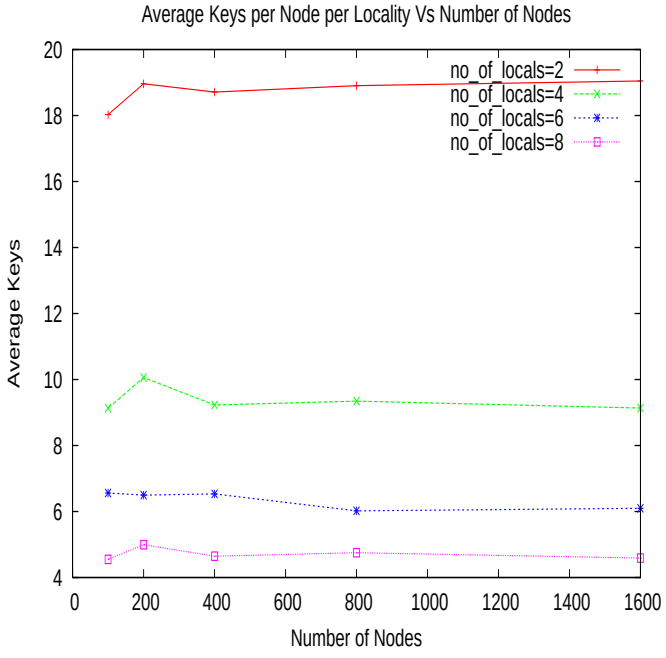
As depicted in the graphs it is evident that the number of datakeys stored decreases with the increase in size of locality. This measurement confirms that L-Chord does not clutter up all the data biased on a single locality. On the otherhand, with the variation in number of nodes the average key distribution per locality per node follows a constant value even with the different size of network which indicates an even distribution across the different nodes. Hence although keys distribution follows a linear path across the localities but load on individual nodes in a locality is evenly distributed (see Figure 5.6(b)).

5.8 Locality Maintenance Overhead

The only extra overhead L-Chord incurs is the maintenance of tag-table. This extra overhead can be minimized if we keep the number of localities across the system low. The normal Internet statistics revealed that the maximum amount of traffic in a P2P system is for the music files (mp3, etc). Hence, the localities for the other categorization can be kept as low as possible. For instance, if we have a locality for the executable programs as "Executables", then increasing sub categorization will lead to increase in tag-table maintenance. Hence, increasing only the number of localities for the popular categorizations, the power of L-Chord can be utilize more specific to the user demands. On an average if we have a Chord ring of 2^m, choosing the number of localities close to the 2^m will defeat the purpose of L-Chord optimization. In this case since, nodes already maintains list of successors

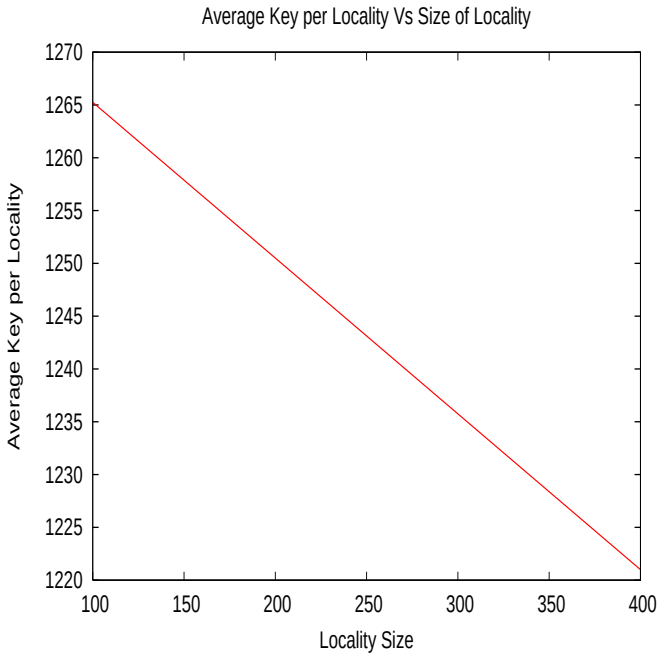


(a) Average Data per Locality Vs Number of Nodes (N)

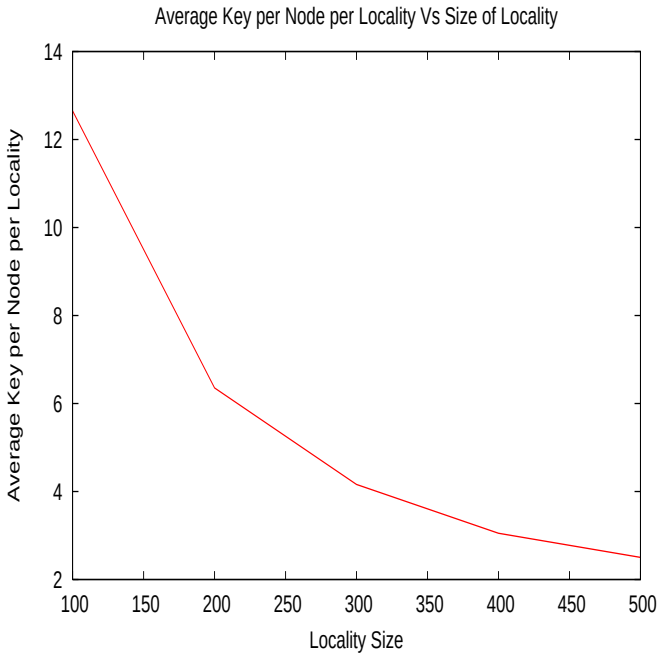


(b) Average Data per Node per Locality Vs Number of Nodes (N)

Figure 5.5: $base = 2$, $localsize = \frac{N}{2}$, $succ = 16$



(a) Average Data per Locality Vs Number of localities



(b) Average Data per Node per Locality Vs Number of localities

Figure 5.6: $base = 2, no\_of\_locals = 4$

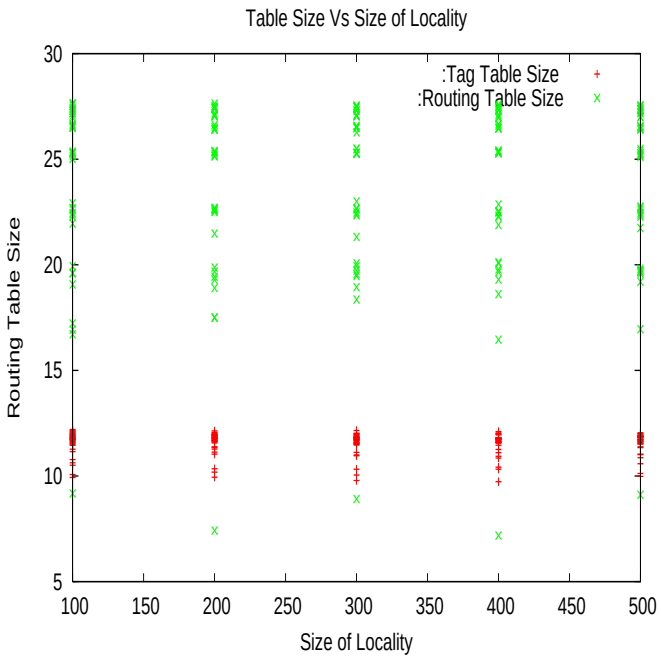


Figure 5.7: Comparison between Tag Table Size and Routing Table Size Vs Locality Size. $base = 2, succs = 16, localsize = 200$

and predecessors, the lookup for tag-node comes with the additional lookup costs for the tag-node. In such a situation, the performance improvement can not be realized. We recommend, the average number of localities should be kept less than or equal to m. This ensures low tag-table overhead mainly because Chord uses finger table for lookups. The size of the finger table is $(b - 1)\ log_b(2^m)$. Keeping number of localities within the limit of m will keep the tag-node within the reach of a node's finger table hence, the extra control messages can be avoided. Figure 5.7 shows a comparison between the Chord's average routing table size and the average tagtable size. As mentioned, the average tagtable size for different locality size is kept low for efficient routing of messages utilizing the finger table entry.

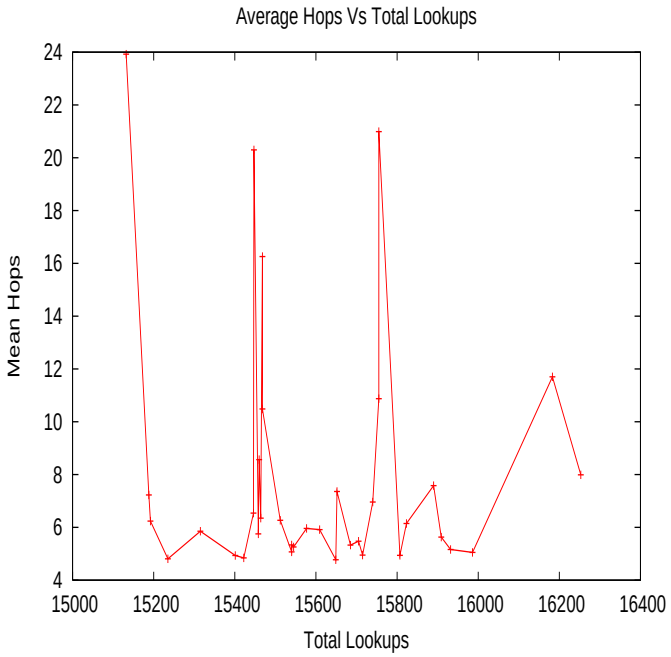
5.9 Comparative Analysis

In this section we will demonstrate the effectiveness of L-Chord over Chord. A comparative result is demonstrated and performance of both the models are evaluated. For both the models separate simulations are performed and logs are collected by varying the stabilization timers. The seed for generation of lookups is kept constant for both the models to ensure number of lookups to be in the near range. Note it is difficult to keep the number of lookups constant because of the randomness in the event generation and other related factors in terms of their join, departure and other behaviors. Also we have observed simulation generates greater number of lookups for Chord as compared to L-Chord for same parameters. This is because the timer for event scheduling is same for both the models. In case of L-Chord, extra lookup for tagnodes consumes the time hence resulting in less number of lookups.

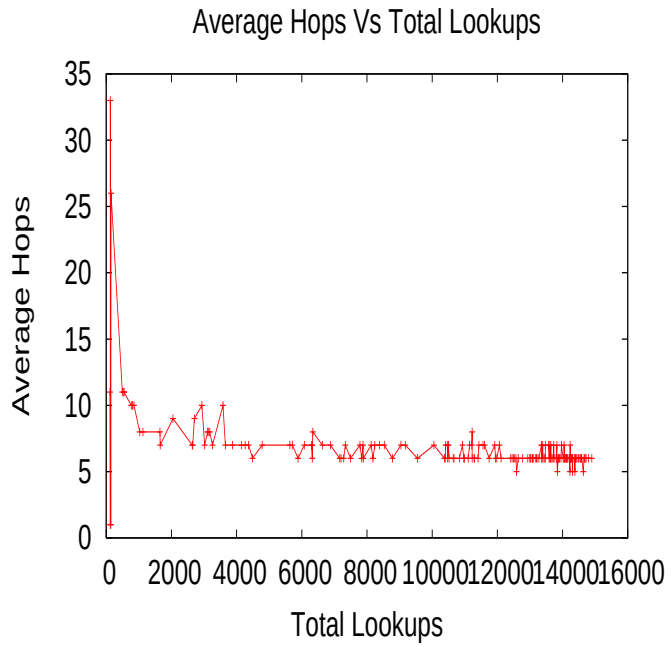
5.9.1 Hops Comparison

Figure 5.8 shows the comparison of L-Chord and Chord with respect to the average hops performance. As can be seen from the graph, in Chord average hops shoots with different lookups. This is because for some lookups the distance between the nodes plays an important role. On the other hand L-Chord average hops maintains a fluctuation between [5 - 8]. In L-Chord, once the system is settled after the realization of all the localities, tagnode lookups greatly reduces the average hops.

It is also observed that for L-Chord, average hops shoots as bad as beyond 30. In L-Chord, we have observed in the worst case, the lookup for tagnode and the node within a locality can both fail. In such a situation the overhead of tagnode lookup further increases the hop cost. Also when large number of nodes crash at the same time, the number of hops increases and large number of lookups are consumed. The



(a) Average Hops Vs Total Lookups. $base = 2, succs = 16$ for Chord



(b) Average Hops Vs Total Lookups. $base = 2, succs = 16, no\_of\_locals = 4$ for L-Chord

Figure 5.8: Hops Comparison between Chord and L-Chord

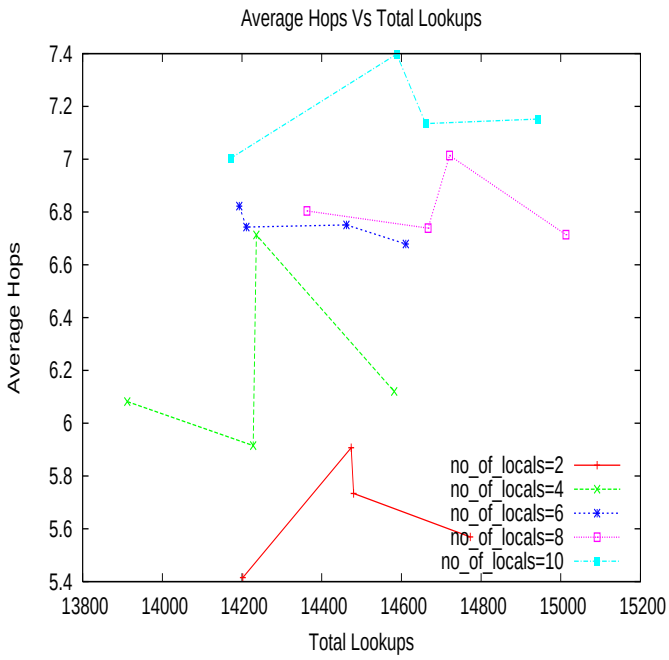


Figure 5.9: L-Chord Average Hop Performance

system waits on the nodes to re-join and there after the stabilization takes additional time to fix the nodes successors and predecessors list. During this process the query propagates across the network to reach a node closer to its key. Since the nodes are crashed so before the stabilization occurs false query propagates due to unstable finger table leads to an additional increase in number of hops. Figure 5.9 shows the average hops comparison for different locality size. It is evident from the graph that the average hops remains low in L-Chord.

Figure 5.10 shows the average hop performance of L-Chord compare with the orignal Chord with variation in number of nodes. Notice the rise in curve is more pronounced in Chord as compare to the L-Chord. For all the localities, the graph shows a lower rise with the increasing number of nodes. As mentioned, in worst case the average hop for L-Chord shoots badly in comparison with the Chord and hence we obtain a higher average hops when compared as a whole.

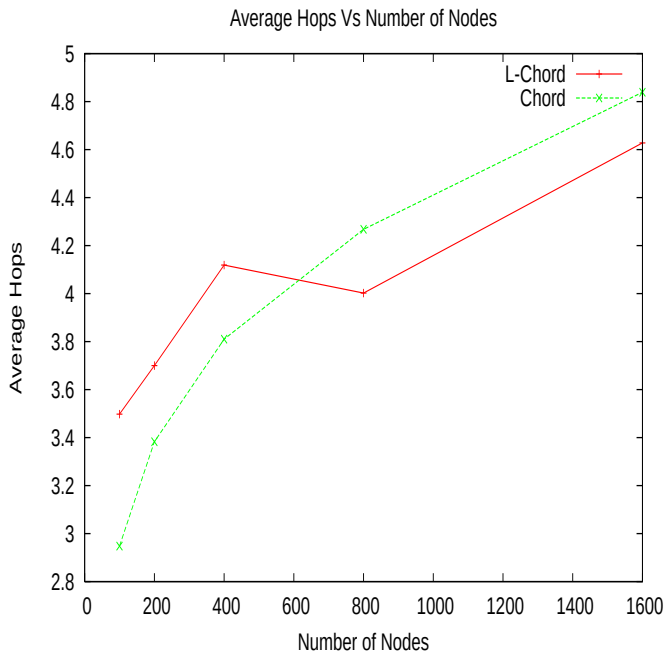
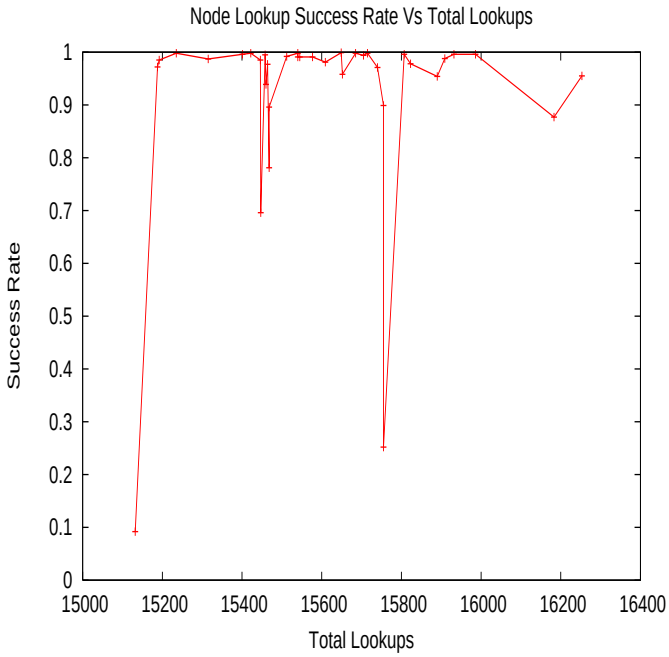


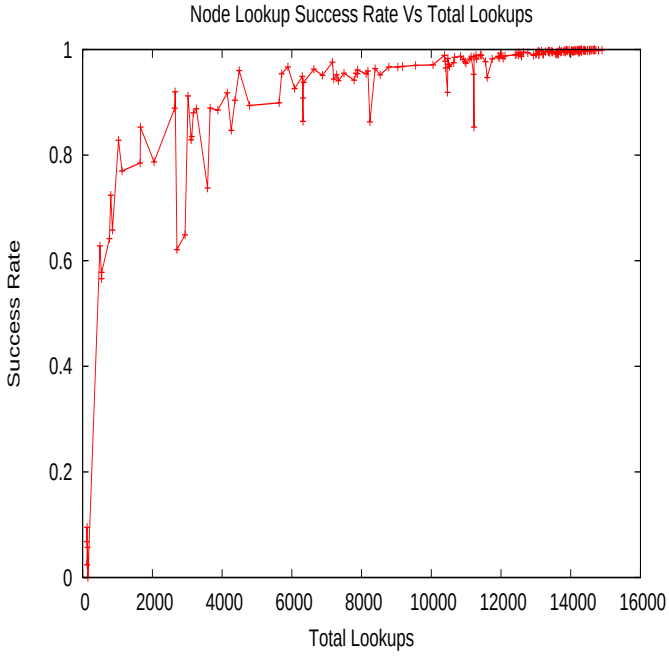
Figure 5.10: Comparison of Chord and L-Chord Average Hop Performance with Variation in Number of Nodes (N)

5.9.2 Success Rates

The success rates is evaluated on the basis of total lookups for a nodes. Total successful lookups for a node is the sum of lookups for the data found successfully inside the node and when data is not found sccessfully inside the node. Figure 5.11 shows the success rate performance for Chord and L-Chord. As can be seen from the graph, in both the cases except for some failed lookups the average performance for both the model closes to 100%. Figure 5.12 confirms the above result with variation in number of nodes.



(a) Success Rate Vs Total Lookups for Chord



(b) Success Rate Vs Total Lookups, $no\_of\_locals = 4$ for L-Chord

Figure 5.11: $base = 2, succs = 16$

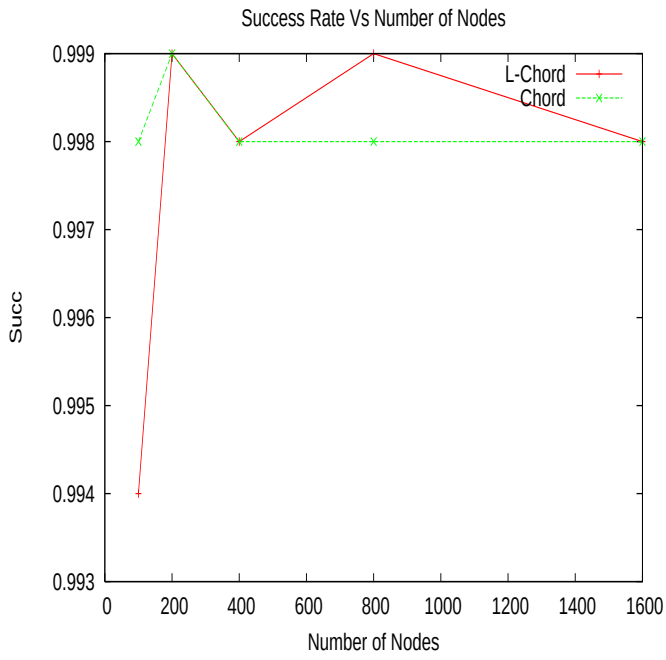


Figure 5.12: Comparison of Chord and L-Chord Success Rate with Variation in Number of Nodes (N), $localsize = \frac{N}{2}$

5.10 Summary

The results demonstrate that the locality size plays an important role in determining the lookup performance of L-Chord. Keeping a tight relation removes the extra hops which leads to low success rate. The evaluation of L-Chord is provided with the evaluation of search performance. However, performance related to other factors such as join, crash can also be evaluated. The aim of this thesis is to identify the locality framework and its impact on search. With the introduction of this philosophy, this work can be extended to evaluate L-Chord performance on the other types of networks not only in context of search but also in terms of other factors which constitutes a complete P2P system. Section 5.9 describes the effectiveness of L-Chord over Chord and achieves a close model similar in lookup success rate and better in avergae hop length for a query to propagate.

Chapter 6

Conclusion

In this work we proposed an improved search mechanism for P2P systems. The locality scheme is defined to model a P2P system as different localities of similar types of data. However, the locality scheme is not limited to Chord or a structured P2P network in general, it can be utilized to improve the search mechanism of other P2P systems as well. Hence we conclude:

1. Locality based approach provides an efficient modelling scheme for a P2P network.

2. A Locality narrows the domain of a search by targeting query to its relevant semantical significance.

3. Locality provides a robust and highly scalable approach to improve search mechanisms.

In this paper we have provided a scheme to impose our locality model over Chord. However, this scheme can also be applied to Chord in other ways. For instance, L-Chord can be remodeled to form different overlays of complete Chord rings connected through a representative node. Also we have not modified the placement of nodes

on the ring. In a different approach we can force peers instead of data to join the localities. We know that a Chord node joins the network with the help of a well known node. If, the well known node plays the part of a locality representative, then separate Chord rings can be realized along the well known node. However, this approach requires maintenance of multiple images of nodes, if a node serves in multiple localities. Also the overhead on the locality representatives will increase.

The simulator is designed with the assumption that the probability of the search key to be present on the network is 50%. As mentioned in the section 5.5.2, L-Chord can also be evaluated with the same Chord philosophy by controling the datakey lookup for the keys always present on the network. This work can be further extended to evaluate the performance with respect to the join and the departure of the nodes as well. Currently, we have observed that multiple node failure leads to unexpected longer hop length. This is because, the simulator does not support looking up multiple successors. Hence, every time for a key lookup only immediate successor is referenced.

The motivation in this work focuses mainly on improving the search mechanism. We have not evaluated the framework on unstructured P2P network. Moreover, the evaluation process is a single threaded simulation. The simulator thread library is an illusion to a programmer as multiple threads. In reality, operating system schedules a single thread and tasks scheduling is cooperative [20]. More realistic results can be generated by developing a multi threaded simulator or using other practical live simulation techniques provided by PlanetLab, et al [21].

Bibliography

[1] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, 2003.

[2] T. Sundsted, "The practice of peer-to-peer computing: Introduction and history." [Online]. Available: http://www-106.ibm.com/developerworks/java/library/j-p2p/

[3] R. Schollmeier, "[16] a definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *P2P '01: Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)*. Washington, DC, USA: IEEE Computer Society, 2001, p. 101.

[4] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Comput. Surv.*, vol. 36, no. 4, pp. 335–371, 2004.

[5] D. Doval and D. O'Mahony, "Overlay networks - a scalable alternative to p2p," in *IEEE internet computing*, August 2003, pp. 2–6.

[6] D. Takemoto, S. Tagashira, and S. Fujita, "Distributed algorithms for balanced zone partitioning in content-addressable networks," in *ICPADS '04: Proceedings of the Parallel and Distributed Systems, Tenth International Conference on (ICPADS'04)*. Washington, DC, USA: IEEE Computer Society, 2004, p. 377.

[7] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM Press, 2001, pp. 131–145.

[8] K. W. Ross and D. Rubenstien, "P2psystems," in *Electronics proceesdings for the 1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, March 2002.

[9] J.-M. Wong, "Gene Kan: How Gnutella Happened," February 2001. [Online]. Available: http://news.dmusic.com/print/3641

[10] "The practice of peer-to-peer computing: Introduction and history." [Online]. Available: http://www.limewire.org/techdocs.shtml

[11] H. Cai and J. Wang, "Foreseer: a novel, locality-aware peer-to-peer system architecture for keyword searches," in *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2004, pp. 38–58.

[12] M. W. Barbosa, M. M. Costa, J. M. Almeida, and V. A. F. Almeida, "Using locality of reference to improve performance of peer-to-peer applications," in *WOSP '04: Proceedings of the fourth international workshop on Software and performance*. New York, NY, USA: ACM Press, 2004, pp. 216–227.

[13] "Peer-to-peer (p2p) and how kazaa works." [Online]. Available: http://www.kazaa.com/us/help/glossary/p2p.htm

[14] B. M. Kunwadee Sripanidkulchai and H. Zhang, "Efficient content location using interest-based locality in peer-to-peer systems," in *INFOCOM 2003: Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, vol. 3. San Francisco, USA: IEEE Computer and Communication Societies, 2003, pp. 2166 – 2176.

[15] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica, "Complex queries in dht-based peer-to-peer networks," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*. London, UK: Springer-Verlag, 2002, pp. 242–259.

[16] T. M. Gill, F. Kaashoek, J. Li, R. Morris, and J. Stribiling, "p2psim - a simulator for peer-to-peer protocols." [Online]. Available: http://pdos.csail.mit.edu/p2psim/

[17] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a dht for low latency and high throughput," in *First Symposium on Networked System Design and Implementation*, San Francisco, CA, March 2004.

[18] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: estimating latency between arbitrary internet end hosts," in *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. New York, NY, USA: ACM Press, 2002, pp. 5–18.

[19] [Online]. Available: http://pdos.csail.mit.edu/p2psim/kingdata/

[20] "Libtask - a coroutine library for c and unix." [Online]. Available: http://swtch.com/libtask

[21] "Planetlab - an open platform for developing, deploying, and accessing palnetary-scale services." [Online]. Available: http://www.planet-lab.org/