Some Novel Approaches to Improve Performance of Peer-to-Peer Systems

A Dissertation Submitted in Partial Fulfillment of the Requirement for the Award of the Degree of

PhD

Submitted by

Guruprasad Khataniar

03610103

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

ASSAM, INDIA - 781039

2010

C E R T I F I C A T E

This is to certify that this thesis entitled **"Some Novel Approaches to Improve Performance of Peer-to-Peer Systems"** *submitted by Guruprasad Khataniar, to the Indian Institute of Technology Guwahati, for partial fulfillment of the award of the degree of Doctor of Philosophy, is a record of bona fide research work carried out by him under my supervision and guidance.*

The thesis, in my opinion, is worthy of consideration for award of the degree of Doctor of Philosophy in accordance with the regulations of the institute. To the best of my knowledge, the results embodied in the thesis have not been submitted to any other university or institute for the award of any other degree or diploma.

Place : I.I.T. Guwahati, India
Date:

(Diganta Goswami)
Associate Professor,
Dept. of Computer Science and Engineering,
Indian Institute of Technology Guwahati.

Acknowledgements

This thesis marks the end of a long educational path and I would like to thank almighty who helped me along the way.

I feel greatly privileged to express my deepest and most sincere gratitude to my supervisor Dr.Diganta Goswami for the invaluable suggestions and guidance during the course of the thesis, without which this work would not have been materialized. His advice during the long Ph.D. process helped me in maintaining my focus and guiding me towards a better future. I really appreciate his encouragement and motivation as well as the freedom he gave to develop my own pace and interests. I am also grateful for his support during my stay at hostel.

I would like to thank Professor Sukumar Nandi for his encouragement to do hard labor. I also thank Dr.J.K.Deka and Dr.S.V.Rao for their valuable suggestions at different stages. I would also like to thank all other faculty members of CSE Department for their kind help in carrying out this work.

I would like to thank staff members of CSE Department for their selfless help during my doctoral period. I feel I need to give special thanks to Nanu, Bhriguraj, Gautam, Raktajit, Souvik, Naba, Prabin and Kulen.

I have learned a lot from countless colleagues at IITG. People who helped deepen my understanding of Peer-to-Peer include Pallab, Biswanath, Brajesh,

Sandeep, Gunakanta, Amrita and Mauchumi are few to earn special thanks.

Last but not least, I would like to express my extreme gratitude to my wife Archana, my son Himsikhar and my daughter Gaargi to whom I dedicate this work. All, you were those who directly felt the pressure and the stress of my everyday life in the long period of this thesis. This work could not have been achieved without your patience, your assistance and your love.

Abstract

Over the years Peer-to-Peer (p2p) has become an accepted paradigm for large-scale Internet applications. A p2p system is defined as an autonomous, self-organized, scalable distributed system with shared resource pool in which all nodes have identical capabilities and responsibilities and all communications are normally symmetric. Peer-to-Peer networking is becoming increasingly popular mostly due to the number of applications running on top of such networks enabling nodes (called peers) to collaborate between each other to perform specific tasks. Early p2p-based file sharing applications employed so-called unstructured approaches which relied on look-up via a central server which stored the locations of all data items. Later approaches, like Gnutella, use a flooding technique where look-up queries are sent to all peers participating in the system until the corresponding data item is discovered. Apparently, neither approach scales well, since in the server-based system the central server becomes a bottleneck with regards to resources such as memory, processing power, and bandwidth; while the flooding-based approaches show tremendous bandwidth consumption on the network. Researchers, later on, developed structured system where the underlying network and the number of peers can grow arbitrarily without impacting the efficiency of the distributed application in marked contrast to unstructured p2p systems. These systems are based on Distributed Hash Tables (DHTs) that provide distributed indexing as well as scalability, reliability, and fault tolerance. Commonly, a data item can be retrieved from the network with a complexity of $O(\log n)$. But these structured systems cannot efficiently handle high churn rate

(high rate of joining and leaving of nodes) in the system since the complexity of handling failure (or joining / leaving) of a node is $O(\log n)^2$.

Extensive applications of p2p systems demand an effective solution for efficient query processing, handling of churn rate, load balancing and maintenance of healthy arrangement of nodes for the improved response of the system. This work mainly focuses on developing mechanisms to improve performance of p2p systems covering areas of structured and unstructured overlay systems. We have found the following two aspects to be important and useful towards achieving this goal: use of hierarchical approaches for constructing the network; and classification of nodes (peers) based on various parameters and placing them in different levels of hierarchy based on the category.

We first develop a structured hierarchical p2p (SHP) system that gives improved performance over existing well known structured system. This architecture is capable of handling high churn rate and found to provide efficient support for two types of queries: point query and range query. We also introduce a concept called 'temporal centralization' in an existing structured overlay to handle high churn rate without disturbing the original decentralized architecture of the system. An organized network architecture for unstructured p2p systems have been developed where nodes are added to the network in a systematic way to efficiently utilize the resources of the nodes. This network architecture, denoted as unstructured hierarchical p2p (HUP), is characterized by $O(\log_m n)$ network diameter and $O(\log_m n)$ messages for node joining / leaving, where n is the number of nodes in the network and m is a constant. Purely decentralized systems like Gnutella route the queries in an environment where the node capabilities are not identified; whereas HUP routes the queries towards the high capable nodes improving the probability of query success rate. We have also proposed an incentive mechanism which encourages users to participate with meaningful information which helps in elimination of free-riders and thus, improving overall performance the system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the years Peer-to-Peer (p2p) systems have gained increased popularity because it can be used to exploit the computing power, resources and storage of a large population of networked computers in a cost-effective manner. A p2p network exploits diverse connectivity among participants in a network and makes efficient use of the collective bandwidth of network users in contrast to conventional centralized resources where a relatively low number of servers provide the core value to a service or application. Although peer-to-peer networking is mostly applied to file-sharing applications like Napster [BYL09], BitTorrent [Coh03], and KaZaA [BYL09][LRW03], a substantial amount of research projects have been launched overtime to exploit the new possibilities provided by this new networking paradigm. Services like video streaming (live and on-demand), Voice-over-IP, etc. and distributed applications like SETI@home [ACK+02] are some of the examples. According to several Internet service providers, more than 50% of Internet traffic is due to p2p users [SE05]. Sandvine [Rep08] presented statistics of Internet traffic in North America in 2008. In aggregate 44% consumer broadband traffic was p2p, a slight increase from 2007 when 40.5% of traffic was p2p; web browsing was 27% and streaming was 15%. Web browsing, p2p, and streaming constituted much of the downstream traffic, while p2p dominated upstream with 75% of the upstream traffic.

The concept of p2p networking is not new and is started in the late 1960s with the starting of networking (ARPANET), which was intended for file sharing among users without the concept of client and server. Every host was being treated equally and one could therefore call this network a p2p network. But, only after 1999, with Napster [BYL09], the modern p2p systems came to the picture. A p2p system normally uses the concept of overlay, a logical layer on top of the physical network, for message delivery between peers. In a client-server system, the server acts as the central hub to provide services to the clients. Server is the central entity and only provider of service and content. Usually servers are supposed to have much higher performance/capacity compared to clients. But in p2p system no distinction is laid down for content provider (server) and content requester (client), where resources are shared between the peers and resources can be accessed directly from other peers. The first generation of p2p networking started with the centralized concept with a central server. However, contrary to the client-server approach this server only stores the indexes of peers where the actual content is available, thus greatly reducing the load of that server. Every peer is connected to the centralized lookup server, to which it can issue requests for content matching the keywords stated in the request. This concept was widely used and became well known due to Napster [BYL09]. No file could be found in Napster if the central lookup table were not available. Only the file retrieval and the storage are decentralized. Thus, the server leads to bottleneck and single point of failure. The computing power and storage capabilities of the central lookup facility grow proportional to the number of users, which also affects the scalability of this approach. Thereafter, came the pure p2p without the concept of central server. The queries were flooded over the network. Gnutella 0.4 [SR05][pv07a] and Freenet [CSWH00] are examples of pure p2p model. The major concern in this model is the high bandwidth consumption due to flooding of queries. An efficient way to reduce the consumption of bandwidth is the introduction of hi-

erarchies. To avoid traffic overloading, schemes like Gnutella 0.6 [pv07b][Rip01], KaZaA [BYL09], JXTA [MM02a] etc. were introduced as second generation of p2p. Some of nodes are treated as special nodes known as superpeers with high capabilities. The superpeer acts as the dynamic central entity for joining and processing queries.

All these above discussed systems use unstructured overlays, which usually exhibit linear search complexity at its best. The unstructured overlays are easy to implement but have inefficient routing and inability to locate rare objects. In a structured overlay nodes are arranged in a systematic way and key *id*s are placed in specific location. It is the most recently stated overlay, where search is deterministic in nature. In a structured system, each data item is assigned an identifier, *id*, a unique value from the address space. Then it stores a file at the node responsible for the portion of the address space which contains the identifier. A data item can be retrieved from the system with a complexity of $O(\log n)$, where n is the number of nodes in the system. The underlying network and the number of peers in structured approaches can grow arbitrarily without impacting the efficiency of the distributed application. Structured systems like Chord [SMLN$^+$03][SMK$^+$01], CAN [RFH$^+$01], Pastry [RD01a], Tapestry [ZKJ01][ZHS$^+$04], P-Grid [Abe01][ACMD$^+$03], etc. employ a globally consistent function to map a file name with key *id*, hence the searching is deterministic. But a structured p2p system assumes that all the peers have same priorities and features, which is infeasible in practice. From Gnutella analysis [ZyF02][AH00] it is found that all the peers in p2p systems do not contribute homogeneously. There may be various computers with heterogeneous specifications, different sharing capacities, variable life-time in the system. Therefore, hierarchical system is introduced to develop a new category of p2p system [BYL09]. The major advantages of a hierarchical system are scalability, efficient lookup, easy maintenance of nodes etc.

In this chapter we have presented the introductory concept of p2p. In section 1.1 we have defined p2p based on the findings of literature survey. Section 1.2 gives a brief introduction of overlay and section 1.3 discusses the distributed hash table which is the main building block of most of the structured system. In section 1.4 we elaborate our motivation and objective of this thesis and finally we have stated the outline of the thesis in section 1.5.

1.1 Defining Peer-to-Peer

The Internet based applications require scalability, security, reliability, flexibility and quality of service. The client-server based applications are unable to satisfy the requirement of Internet as it is growing exponentially in terms of users and resources. Peer-to-Peer has emerged as a new paradigm for communication on the Internet that can address some of these issues. Different definitions have been found in literature based on system architecture and configurations. Some of these are stated below.

- A distributed network architecture may be called a Peer-to-Peer (p2p) network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers, etc.). These shared resources are necessary to provide the service and content offered by the network (e.g. file sharing or shared workspaces for collaboration). They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (service and content) providers as well as resource requesters (servent-concept) [Sch01].

- Peer-to-peer systems and applications are distributed systems without any centralized control or hierarchical organization, where the software running at each node is equivalent in functionality. A review of the features of p2p applications yields a long list: redundant storage, permanence, selec-

tion of nearby servers, anonymity, search, authentication, and hierarchical naming [SMK+01].

- Peer-to-peer systems can be characterized as distributed systems in which all nodes have identical capabilities and responsibilities and all communication is symmetric [RD01a].
- Peer-to-Peer networking refers to a class of systems, applications and architectures that employ distributed resources to perform routing and networking tasks in a decentralized and self organizing way [ZSKT05].
- Peer-to-Peer systems are distributed systems consisting of interconnected nodes able to self organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority [LCP+05].
- A p2p is a self-organizing system of equal, autonomous entities aims for the shared usage of distributed resources in a networked environment avoiding central services [SE05].

From the above one can define P2P as an *autonomous, self-organized, scalable distributed system with shared resource pool without a single point of failure in which all nodes have identical capabilities and responsibilities and all communications are normally symmetric; and where main characteristics of the participants are decentralized resource usage and decentralized self-organization.*

1.2 Overlay System

A p2p overlay is a distributed collection of autonomous end system computing devices called peers that form a set of interconnections called an overlay to share

resources of the peers such that peers have symmetric roles in the overlay for both message routing and resource sharing. The characteristics of an overlay can be summarized as below.

- The peers self-organize the overlay.
- The peers have symmetric roles.
- Peer-to-Peer overlays are highly scalable.
- Peers are autonomous.
- A p2p overlay provides a shared resource pool.
- Peer-to-Peer overlays are resilient in the face of dynamic peer membership, referred to as churn.

An overlay is a logical abstraction of the physical (underlying) network needed by distributed applications. The nodes for the overlay network and connections are decided by the application. The routing algorithm used in underlying network is different from that used by overlay network. Figure 1.1 illustrates the relation between the overlay and the underlay. Here if peer 'A' sends information to peer 'C'. Then actual data transfer occurs from 'a' to 'g'. Hence for a p2p, user actual data transfer is transparent in its application.

The overlay is a logical layer for message delivery between peers. If two peers are not directly connected, they can use the overlay routing mechanism to send messages to each other indirectly via other peers. There is a wide range of schemes to organize the overlay. These schemes vary by how much state each peer has to keep, how much message overhead is needed to stabilize the overlay and the message delivery performance that the overlay provides.

Peers are autonomous and may leave the overlay at any time. New peers may join the overlay. Fluctuations in overlay membership may ripple through

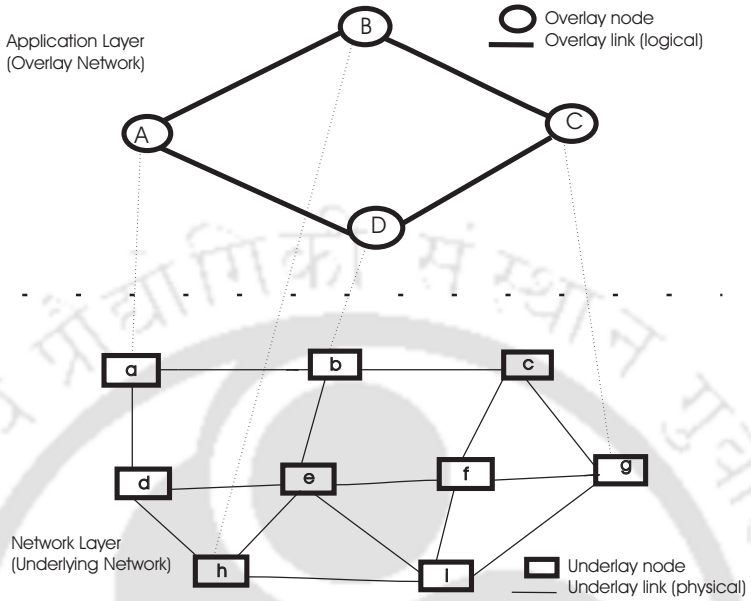

Figure 1.1: Illustrating overlay and underlay

the overlay, changing overlay peer relationships. For example, peers may change their adjacencies and neighbors in the overlay due to membership changes. The coordinated management of overlay routing state among the peers is called overlay maintenance. Since the peers are collectively coordinating the management of the overlay routing behavior without the use of a centralized overlay manager, p2p overlays are self-organizing.

A p2p overlay can be viewed as a directed graph $G = (V, E)$, where V is the set of nodes in the overlay and E is the set of links between nodes. Nodes are located in a physical network, which provides reliable message transport between nodes. Each node p has a unique identification number *pid* and a network address *nid*. An edge (p, q) in E means that p has a direct path to send a message to q; that is, p can send a message to q over the network using *nid* of q as the destination. It is desirable that G be a connected graph. Maintaining connectedness and a consistent view of G across all nodes is the job of the overlay maintenance

mechanism. Due to peer's joining and leaving the overlay, the overlay graph G is dynamic.

1.3 Distributed Hash Table

The concept of Distributed Hash Table (DHT) is important for discussion of structured p2p overlays. Therefore, we have given an overview of the DHT in this section. According to Lynch et al. [LMR02] the goal of a DHT is to support three operations: join, leave and data update. Joins and leaves are initiated by nodes when they enter and leave the service, respectively. An update is a targeted request initiated at a node and is forwarded toward its target by a series of update-step requests between nodes.

A DHT manages data by distributing it across a number of nodes and implementing a routing scheme which allows one to efficiently look up the node on which a specific data item is located. DHTs are a technique developed to store data over p2p networks. They are intended to be used as a primitive layer providing a lookup service to a variety of applications. Their function is analogous to that of a standard hash table, whereby data is stored in $< key, value >$ pairs, but instead of the table being stored locally on one machine, its contents are spread around the members of the DHT network. Thus, to lookup the value stored at a particular key in a DHT, first it is necessary to ascertain which node is responsible for that key and then to perform a standard hash table lookup at that node. The key used for the DHT is typically a long bit sequence produced by applying an encryption algorithm, such as MD5 (128-bit) or SHA-1 (160-bit), to the content to be stored as the value in the table.

These approaches are also often called structured p2p systems because of their structured and proactive procedures. DHTs provide a global view of data distributed among many nodes, independent of the actual location. Thereby,

location of data depends on the current DHT state and not intrinsically on the data. The characteristics of DHTs can be summarized as follows.

- In contrast to unstructured p2p systems, each DHT node manages a small number of references to other nodes. By means there are $O(\log n)$ references, where n depicts the number of nodes in the system.
- By mapping nodes and data items into a common address space, routing to a node leads to the data items for which a certain node is responsible.
- Queries are routed via a small number of nodes to the target node. Because of the small set of references each node manages, a data item can be located by routing via $O(\log n)$ hops. The initial node of a lookup request may be any node of the DHT.
- By distributing the identifiers of nodes and data items nearly equally throughout the system, the load for retrieving items should be balanced equally among all nodes.
- Because no node plays a distinct role within the system, the formation of hot spots or bottlenecks can be avoided. Also, the departure or dedicated elimination of a node should have no considerable effects on the functionality of a DHT. Therefore, DHTs are considered to be very robust against random failures and attacks.
- A distributed index provides a definitive answer about results. If a data item is stored in the system, the DHT guarantees that the data is found.

As a storage system, a DHT implements an interface for persistently storing and reliably retrieving data in a distributed manner. On each node, the application interface provides the two main primitives of a hash table. The put primitive takes a $< key, value >$ pair and stores it on the node responsible for the identifier key. Similarly, the get primitive accepts an identifier and returns the value associated with the specified identifier. The interface of a DHT as given in [SE05] is

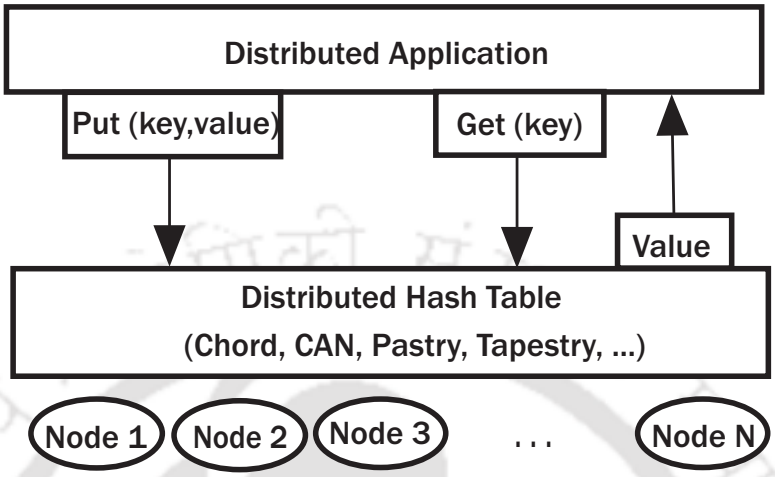

Figure 1.2: Illustrating Distributed Hash Table

shown in Figure 1.2.

DHT provides infrastructure for building different systems like cooperative file system (CFS) [DKK+01], OceanStore [KBC+00], PAST [RD01b], name services [CMM02] etc.

1.4 Motivation and Objectives

Peer-to-Peer computing is attracting increasing attention. Many p2p systems have rapidly become basic platforms for users to share information and resources over the Internet. In essence, a pure p2p system can be described as a decentralized network system in which all participants have symmetric duties and responsibilities. Key characteristics of p2p systems are decentralization, self-organization, dynamism and fault-tolerance, all of which make p2p paradigm very attractive for information storage and retrieval.

Some of the major challenges in designing a p2p system which have motivated our research are as follows.

- Frequent joining/leaving of nodes to/from the system: This problem arises due to transient node population in a system. One possible reason of this problem is due to free riders. A free-rider is a peer that uses a file-sharing application to access content from others but does not contribute content at the same degree.

- Improvement of lookup efficiency: Most of the unstructured p2p systems use some variants of flooding-based scheme for lookup and hence incurs a large overhead. The success rate of query execution in an unstructured overlay decreases as one tries to limit the number of messages by imposing short TTL value. Structured overlays, on the other hand, routes the queries deterministically to achieve success in bounded hops. But these systems suffers from degradation of system performance during high churn rate.

- Execution of point query and range query in a single system with improved lookup efficiency: Efficient execution of both range query and point query in a single system is another issue that is difficult to handle. Existing systems handle these two problems separately. A solution to this issue will inspire people to build new applications using these systems.

- Support for an improved load balancing mechanism: Every peer in a p2p system is supposed to behave identically and share same load. It is very much necessary to design an appropriate load balancing mechanism to prove the performance of the system and to exploit the system to its full capacity.

- Controlled free-riding and white-washing problem with incentive mechanism: A p2p system may collapse totally if the problem of free-riding and white-washing are not controlled properly. Developing a system which will give incentive to the user in a better way is always inspired. An appropriately designed incentive mechanism may be used to control the problem of free-riding and white-washing.

- Proper security schemes for the p2p system: The P2P paradigm generates a vast array of new security concerns. In transitioning from a client-server to a P2P deployment, the core of the service shifts from a set of servers, which are under direct control of the service provider, to peers distributed around the world and operated by millions of independent users. This loss of control leads to a wide range of new vulnerabilities that can be exploited by attackers.

Our aim is to propose solutions to some of the above discussed problems considering both structured and unstructured overlays in different contexts.

As the system size increases, the weak peers can seriously compromise the scalability of the whole network, resulting in inefficient group communication. This is especially visible on bandwidth multi-cast services. To tackle this flaw, the traditional approach revolves around organizing the nodes into a multi-layer architecture. By partitioning the flat network into smaller regions, the system complexity reduces as well. In the context of p2p, a hierarchical overlay consists of multi-tier overlays whereby the nodes are organized into disjoint groups. The overlay routing to the target group is done through an inter-group overlay; then intra-group overlay is used to route to the target node. The hierarchical overlay architecture offers important advantages over the flat overlay as it reduces the average number of peer hops in a lookup and hence minimizes the query latency. The system becomes more scalable and can offer improved performance. Therefore, our proposed systems are motivated by hierarchical nature of organization. Hierarchies in both structured and unstructured overlays are necessary to improve overall performances.

It is detrimental to the performance of a system if all the nodes are considered identical because nodes in a network may differ in terms of available bandwidth, processing power, stability, privacy and security, reliability, amount of data shared

and storage capacity. This aspect is motivated by the Gnutella analysis [AH00], which has shown that 70% of the Gnutella users share no files and 90% of users do not response to the query. Another Gnutella analysis [ZSR06] has shown that 7% peers share more files than all other peers and 47% queries are responded by the top 1% peers. Thus, instead of considering all the peers to be identical, it is clearly necessary to classify the peers into groups based on their performances and other parameters.

Free riding and white-washing are two severe problems faced by the developers of p2p system. Free riders are the temporary peers in the p2p network, which do not contribute any resources yet consume resources freely from the network. Free riders create many problems in the p2p network, such as dropping queries made by others and passing through it, which may degrade the performance of p2p network. Free riders contain either zero contents or some fake contents. A huge number of free riders and very less contributors, may lead to saturation of bandwidth for execution of queries. As growing number of peers become free riders, the system may start losing its p2p spirit and becomes a simple traditional client server system. Many p2p networking systems make provision for imposing penalties over the free riders. Some of such penalties may include not executing requests from the suspected peers, suspend them for some amount of time, permanently de-linking from the network etc. The free riders are also well aware of it. So they play the trick of white-washing. In this technique the free riders re-appear with a different identity to get rid of the penalty imposed by the network. p2p system is based on the contribution made by the peers present in the network. Normally peers are reluctant to contribute their resources due to numbers of reasons like: cost of bandwidth, security threat due to open up of several ports, slowing down of self downloading process. Even though many schemes have been proposed in literature to handle this problem, most of these are found to be not very effective and some are found to be very complex.

1.4.1 Major Objectives of Our Work

The primary goal of this work is to propose some solutions to improve performance of p2p systems. To address this broad objective and with the factors of motivation as discussed above we have set our objectives to improve performance of p2p systems in diversified areas. We have considered both structured and unstructured systems and attempted to provide some solutions towards improving their performances. The major objectives of our work are stated as below.

- It is necessary to introduce hierarchies in a p2p system to improve the overall performance. Hence, our aim will be to design and develop appropriate hierarchical overlays for both structured and unstructured systems. We will first consider structured systems and look for a structured hierarchical overlay protocol that gives better performance. Then we will address the same issue for unstructured systems as well.

- We would attempt to classify nodes for different systems with different nomenclatures to assign proper functionalities. As an example, it is not justifiable to place a high-end server in the same group with a mobile phone. If the nodes are placed in different hierarchies according to their performance, accessibility and availability property, it might help improve the overall performance of the system.

- One of our major objectives will be to develop a system that can handle both point query and range query in an efficient way. Most of the existing systems considers only point query. Many new applications may be developed using this system once this system is in place.

- Handling high churn rate is an important issue in a p2p system as such a system is very dynamic and in many applications the system experience high rate of joining and leaving of nodes. Existing structured systems are not capable of handling this issue efficiently. Hence, we would attempt to

propose a solution to this problem in one of the most important and popular structured system.

- We would propose a new incentive mechanism to control free riders effectively. This new incentive mechanism should encourage users to contribute useful information in a p2p system which in turn may help in reducing the number of free riders.

1.5 Organization of the Thesis

The dissertation is organized as follows.

Chapter 1: This chapter introduces the basic concepts of p2p systems and explains the motivation and objectives of our work.

Chapter 2: This chapter gives a survey of various existing p2p systems reported in the literature. A classification of p2p system is first introduced and different p2p systems under each category are discussed.

Chapter 3: In this chapter, we propose a new Structured Hierarchical Overlay Protocol. This new system can efficiently handle both point and range queries. It is shown that this system improves the performance in terms of execution of queries and joining/leaving of nodes.

Chapter 4: Efficient handling of churn rate in an existing structured system namely, Chord is the main focus of this chapter. We introduce the concept of temporal centralization by giving a suitable node classification. This eventually helps reduce the number of messages substantially during joining and leaving of nodes.

Chapter 5: This chapter presents an organized network architecture for unstructured hierarchical p2p systems. Nodes are added to the network in a systematic way to efficiently utilize the node's resources. It is shown that this new system reduces the number of unnecessary messages and increases the success rate during processing a query.

Chapter 6: We propose an incentive mechanism that is aimed at eliminating free-riders and white-washing problem. The scheme introduces better incentive mechanism with an effective grading system of nodes at different hierarchies.

Chapter 7: Finally, this dissertation concludes with Chapter 7. In this chapter, we summarize the major contributions of our work and critically examine our achievements. We also discuss some of the possible future scopes of our work.

Chapter 2

Literature Survey

The survey of literature on p2p systems is presented in this chapter. The chapter starts with a general introduction to p2p systems. In Section 2.1, a classification of p2p systems is presented based on a taxonomy of overlay networks which is adopted based on the literature survey.

Lua et al. [LCP+05] presents a comprehensive survey on p2p overlay network where p2p overlay networks have been classified into two types: *structured* and *unstructured*. According to them the meaning of structured is that the p2p overlay network topology is tightly controlled and content is placed not at random peers, but at specified locations that make subsequent queries more efficient. The meaning of unstructured p2p is described as the system which is composed of peers joining the network with some loose rules and without any prior knowledge of the topology. The network uses flooding as the mechanism to send queries across the overlay with a limited scope. When a peer receives a flood query, it sends a list of all contents matching the query to the originating peer. They have discussed structured and unstructured p2p overlay schemes on the basis of decentralization, architecture, lookup protocol, system parameters, routing performance, routing state, peer's join and leave, security and reliability and fault resiliency. Stephanons and Diomidis [TS04] present a detailed survey on p2p content dis-

tribution technologies. They have classified p2p applications as communication and collaboration, distributed computation, Internet service support, database systems and content distribution. The p2p content distribution system creates a distributed storage medium that allows for the publishing, searching and retrieval of files by members of its network. As systems become more sophisticated non functional features may be provided, including provisions for security, anonymity, fairness, increased scalability and performance, as well resource management and organization capabilities. In the context of p2p content distribution, the p2p technologies were grouped as p2p applications, p2p content publishing and storage systems, p2p infrastructures, routing and location, anonymity and reputation management. To discuss the overlay network, they have distinguished overlays in terms of centralization and structure. According to the centralization the overlays are purely decentralized architectures, partially centralized architectures and hybrid decentralized architectures. In purely decentralized architectures, all nodes in the network perform exactly the same task, act both as servers and clients and there is no central coordination of their activities. They termed nodes of such networks as "servents" (SERVers + cliENTS). In partially centralized architectures, some of the nodes acting as local central indexes for files shared by local peers, these supernodes are assigned dynamically to avoid a single point of failure. In hybrid decentralized architecture there is a central server facilitating the interactions between peers by maintaining directories of meta-data, describing the shared files stored by the peer nodes. Although the end to end interactions and file exchanges may take place directly between two peer nodes, the central servers facilitate these interactions by performing the lookups and identifying the nodes storing the files. According to structure they have categorized overlays as structured, unstructured and loosely structured. Again the unstructured overlays are divided into hybrid decentralized, purely decentralized and partially centralized. Sung et al. [BAH+06] discuss data management in p2p systems with reference to data location, query processing, data integration and data consistency. The index

management for data in structured and unstructured systems are described.

2.1 Classification of Peer-to-Peer Systems

The p2p systems are classified according to logical arrangement of the nodes. The prime themes for classification of overlays are the relationships of overlay nodes and the topology used. Broadly two categories are proposed in different literatures: unstructured and structured. Ranjan et al. [RHB08] investigate various decentralized resource discovery techniques primarily driven by p2p network model in 2006. They present a summary of grid resource discovery, resource taxonomy with focus on computational grid paradigm, p2p taxonomy with focus on extending the structured systems for indexing d-dimensional grid resource queries and classification of the surveyed approaches based on the proposed p2p taxonomy. The classification of p2p systems is as follows.

- Unstructured: deterministic and non-deterministic
- Structured: DHT-based and non-DHT-based
- Hybrid and
- Hierarchical

We now provide a brief survey of each category with discussion of some representative systems in each category. The systems are compared with respect to desirable features of p2p. Performance of p2p system is based on certain features such as self-organization, distributed control, role of symmetry for nodes, anonymity, naming mechanism and security. For example, a system with good performance for large combined storage, CPU power and resources should have self-organizing feature. The comparison of different unstructured p2p systems are discussed in Table 2.1 based on the characteristics such as decentralization, architecture, lookup protocol, system parameters, routing performance, routing state and peers join and leave.

2.1.1 Unstructured Deterministic Systems

Deterministic systems are those where a look-up operation is successful within predefined bounds. Systems including Napster [BYL09], BitTorrent [Coh03], JXTA [MM02a] fall into this category. In these systems, the object lookup operation is centralized while download is decentralized.

1. **Napster:** This network is the first really large-scale p2p infrastructure deployed over the World Wide Web. In fact, the Napster network is based on a centralized server, with which each network participant is directly connected. Figure 2.1 illustrate the architecture of Napster. After joining the Napster network, a node publishes its resources on the central directory server. The centralized server is aware of all available network resources after the update process. When looking for a resource in the network, a node sends a query to the central directory server which returns the address of one or many providers of the requested resource. Disk space for files in Napster is provided by the clients. One file may be stored in multiple locations. To download a file, a client node first obtains a list of possible locations with desired file from an index server. Then the client user selects the location for the file he or she would like to download, using the supplementary information provided by the index server. The client node then tries to retrieve the desired file directly from the peer node of the selected location. Since the user can select the nearest location with the desired file, they can download files fast. The servers serve every file search. This kind of centralized p2p architectures have serious scalability problems.

2. **BitTorrent:** It is a popular p2p application for distributing very large files to a large group of users. Unlike most p2p systems, which form one large overlay, BitTorrent has a distinct overlay for each file. To download a file, peers exchange different blocks of the content until each peer has the

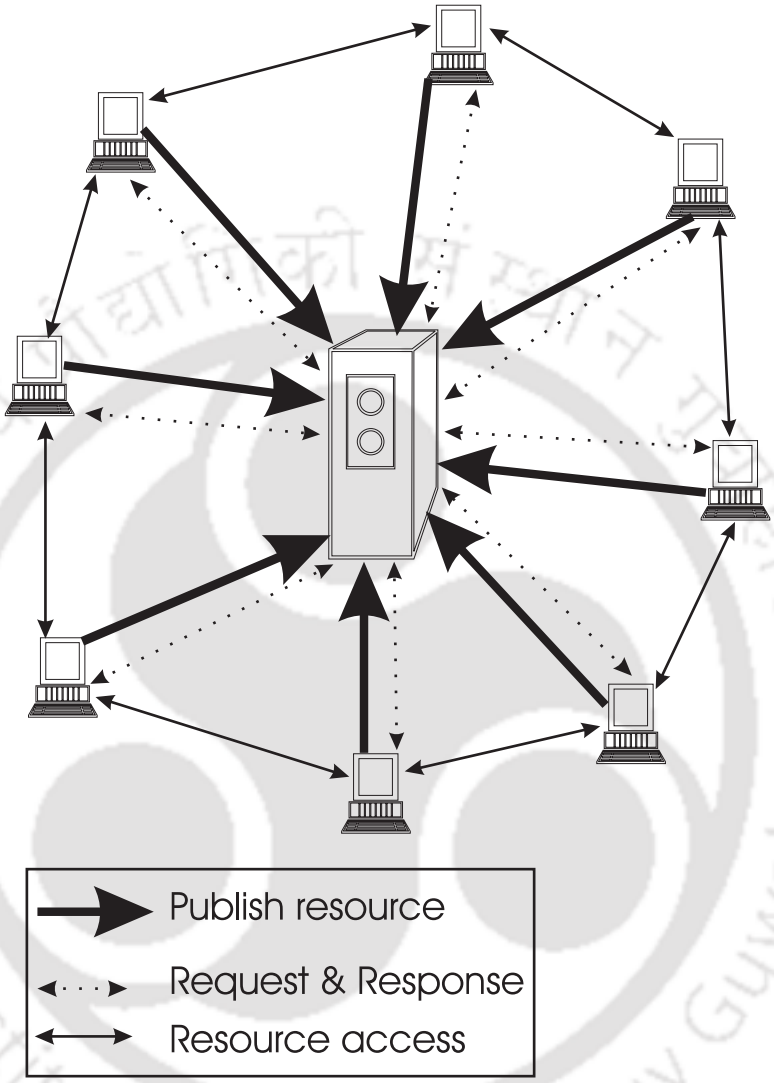

Figure 2.1: Illustrating Napster architecture

entire file. The peers locate one another using a rendezvous point, called a tracker, whose address is provided to new members out of band. Each new peer contacts the tracker via HTTP, periodically sends an update of its progress to the tracker, and informs the tracker when it departs. Each peer may receive the entire file across multiple sessions, i.e., it may obtain only a subset of blocks in one session and resume the download later. Many peers may give up without downloading the whole file. The tracker logs its

interactions with peers, providing the arrival and departure times of peers with one second resolution. While the tracker records the arrival time of all peers, the departure time is only captured for peers which depart gracefully.

3. **eDonkey2000 [HKF+06]:** The eDonkey network (also well known for one of its open-source clients, Emule) was one of the most popular peer-to-peer file sharing networks. The eDonkey provides advanced features, such as search based on file meta-data, concurrent downloads of a file from different sources, partial sharing of downloads and corruption detection. The architecture of the network is hybrid - the first tier is composed of servers, in charge of indexing files and the second tier consists in clients downloading and uploading files.

4. **JXTA:** The JXTA technology is a set of open protocols that enable any connected device on the network, ranging from cell phones and wireless PDAs to PCs and servers, to communicate and collaborate in a p2p manner. The JXTA peers create a virtual network where any peer can interact with other peers and resources directly, even when some of the peers and resources are behind firewalls and network address translations (NATs) or on different network transports.

2.1.2 Unstructured Non-deterministic Systems

In a non-deterministic system the query may not be successful even the required data is present in the system. The systems including Gnutella [SR05], Freenet [CSWH00], FastTrack [LKR06] and KaZaA [BYL09][LRW03] offer non-deterministic query performance.

1. **Gnutella:** It is a protocol for distributed search. Gnutella supports both client-server and p2p paradigms. But its popularity is due to its support for p2p. Gnutella is a distributed software project aiming at the creation

of a true p2p file sharing network without a centralized server. Nodes in Gnutella broadcast requests to their neighbors and use the TTL (Time-to-Live) flag to limit the scope of the request and hence spare bandwidth. When the TTL value is optimally chosen, an existing resource can be found with high probability. However, flooding-based request propagation is indeed robust but impracticable in large-scale p2p systems. With blind flooding, each request generates a very large load for each peer with a complexity of the magnitude order $O(n^t)$, where the parameter n is the number of routing entries per node and t is the TTL value. This load exponentially increases with the TTL and with the rarity of the requested resource.

2. **Freenet:** The key attribute that distinguishes Freenet from other content-sharing systems is its support for anonymity between users. Files uploaded to the network are stored in an encrypted form and replicated throughout the network. Each Freenet member designates an area of their own local storage to contribute to the network, and has no control over what content the network decides to store on their machine. Freenet does not have any built-in search mechanism; resource discovery is achieved via Freenet-hosted discussion boards and directory sites.

3. **FastTrack:** It is a p2p protocol which is used by KaZaA, Grokster [Ref10b], and iMesh [Ref10c] file sharing programs. As of early 2003, the FastTrack was the most popular file sharing network, being mainly used for the exchange of music .mp3 files. It has more users than Napster had at its peak. Popular features of FastTrack are the ability to resume interrupted downloads and to simultaneously download segments of one file from multiple peers.

4. **KaZaA:** It employs the two-tier hierarchical structure with the superpeers indexing files in their managed groups. Figure 2.2 illustrate the architecture of KaZaA. Supernodes serve as referral nodes for files. Supernodes register

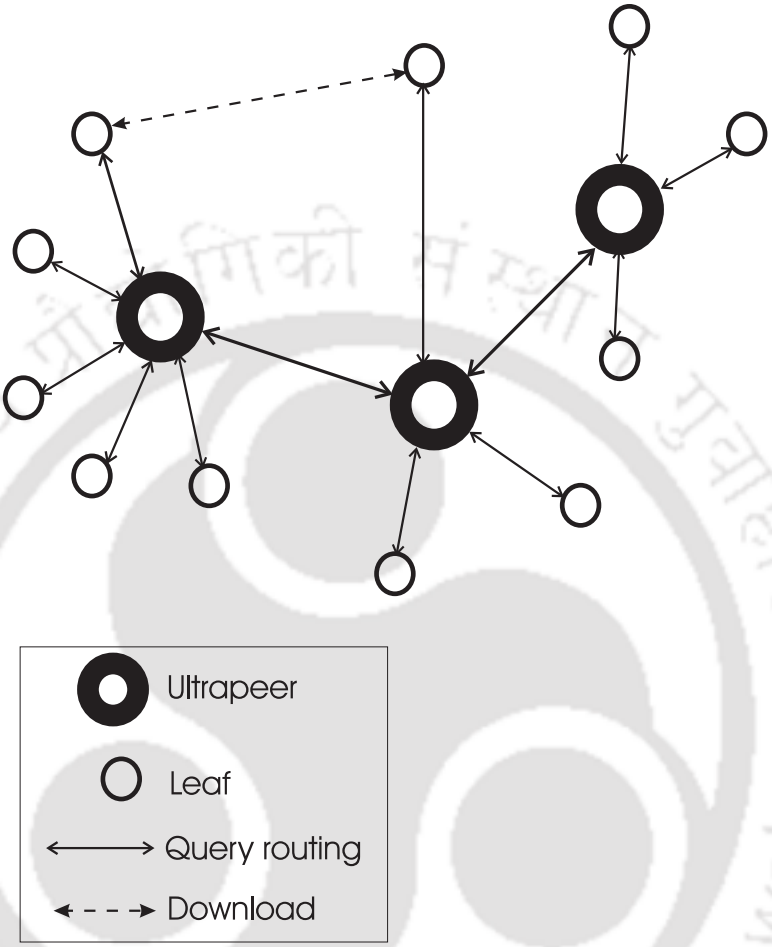

Figure 2.2: Illustrating KaZaA architecture

the list of cached files on those ordinary nodes with which the supernode is connected. An ordinary node A connected to supernode $S1$ communicates with $S1$ to issue requests. $S1$ uses its local database and collaborates with other supernodes to compile a list of other ordinary nodes that store the file and sends this list back to A. A then communicates directly with those ordinary nodes that are on the list to request the content blocks/files they desire. Delivery of the content blocks/files occurs thereafter. At last, A informs $S1$ of new availability of the file it just acquired once the entire file is obtained.

	Freenet	Gnutella	KaZaA	BitTorrent	eDonkey2000
Decentralization	Loosely DHT	Topology is flat with equal peers	Peers are connected to their Superpeers	Centralized model where tracker keeps track of peers	Hybrid 2-layer network
Architecture	Keywords and descriptive text strings to identify data objects	Flat and Ad-Hoc network of peers. Flooding request and peers download directly	Two-level hierarchical network	Peers request information from a central Tracker	Servers provide the locations of files to requesting clients for download directly
Lookup Protocol	Keys, descriptive text string search from peer to peer	Query flooding	Superpeers	Tracker	Client-Server peers
System Parameters	None	None	None	.torrent file	None
Routing performance	Guarantee to locate data using Hops-To-Live (HTL) limits	No guarantee to locate data;Good performance for popular content	Some degree of guarantee to locate data; Good performance for popular content	Guarantee to locate data and guarantee performance for popular content	Guarantee to locate data and guarantee performance for popular content
Routing State	Constant	Constant	Constant	Constant but choking (temporary refusal to upload) may occur	Constant
Peers Join/Leave	Constant	Constant	Constant	Constant	Constant with bootstrapping from other peer and connect to server to register files being shared

Table 2.1: Comparison of unstructured P2P protocols

2.1.3 Structured DHT-based Systems

Structured systems such as offer deterministic query search results within logarithmic bounds on network message complexity. Peers in DHTs maintain an index for $O(log(n))$ peers, where n is the total number of peers in the system, e.g. Chord [SMK+01] [SMLN+03], CAN [RFH+01], Pastry [RD01a], Tapestry [ZKJ01] [ZHS+04], P-Grid [Abe01] [ACMD+03] . The comparison of different structured p2p systems are discussed in Table 2.2 based on the characteristics such as decentralisation, architecture, lookup protocol, system parameters, routing performance, routing state and peers join and leave.

1. **Chord:** It is an example of a distributed hash table (DHT). The nodes are arranged from identifier 0 to $2^m - 1$ on an identifier ring, where m is the number of bits of an identifier. The hashing function is assumed to be consistent, meaning the nodes are evenly distributed over the entire range. An object that gets hashed to key k is handled by $successor(k)$, which is the first node with identifier greater than or equal to k mod 2^m. Each node maintains a routing table (Finger Table) with $\log n$ entries, where n is the total number of nodes in the system. The i^{th} entry in the finger table of node P contains the identifier of the node which exceeds P's identifier by $2^i - 1$.

   For routing a query on key id at node P if id lies between P and its successor, the id would reside at the successor and the address of successor is returned. If id lies beyond the successor, then node P searches through the m entries in its finger table to identify the node Q such that Q most immediately precedes id, among all the entries in the finger table. As Q is the closest known node that precedes id, Q is most likely to have the most information on locating id i.e., locating the immediate successor node to which id has been mapped.

   For join operation, Chord performs three tasks to preserve the steady state:

initializing the finger table of new node, updating the finger tables of existing nodes, and notifying neighbors of new node in order to transfer responsibility of object references. The entire process requires $O(\log^2 n)$ messages. The leaving process also requires the same number of messages to stabilize the network. Although Chord provides the guarantee that a stored data item will be always retrieved, it performs poorly when the churn rate is high.

2. **CAN:** It is another distributed and structured p2p lookup service. Each key is evenly hashed into a point of d-dimensional space as its identifier. When a node joins, it randomly selects a point of d-dimensional space. Then it is responsible for half of regions this point belongs to, and hold all keys whose ids belong to this region. For example, the first arrival node $n1$ is responsible for the whole space/region, the second arrival node $n2$ splits whole region into two parts and takes one of them, and the third arrival node $n3$ splits the $n1$ region (if the random point that it selects belongs to this region) or $n2$ region (otherwise) into two parts and takes one of them also. Each node keeps its neighbor node id locally, and routing is then performed by forwarding requests to the regions closest to the position of the key. In this way, the expected search length (or p2p hops) is $O(d\sqrt[d]{N})$, and state information kept locally is $O(d)$.

3. **Pastry:** This is an alternative approach to implementing a DHT, with similar objectives to CAN, i.e. scalability, self-maintenance, minimized latency and fault-tolerance. Again, it uses the concept of nodes within the network being responsible for specific keys. Nodes joining a network are assigned an id which places them at a point in a circular space covering the numerical range $[0-2^{128})$. Node ids and keys are treated as digits with base 2^b , where b is a system parameter typically set to 4. Messages are routed to the node with id numerically closest to the specified key.

4. **Tapestry:** It is an overlay infrastructure based on Plaxton [PRR97] location and routing mechanism. Each node has a neighbor map, which is organized into routing levels, and each level contains entries that point to a set of nodes closest in network distance that matches the suffix for that level. Each node also maintains a back pointer list that points to nodes where it is referred as a neighbor. They are used in node integration algorithm to generate neighbor maps for a node.

5. **Kademlia [MM02b]:** It is a p2p storage and lookup system. It minimizes the number of configuration messages nodes must send to learn about each other. Configuration information spreads automatically as a side-effect of key lookups. Nodes have enough knowledge and flexibility to route queries through low-latency paths. Each Kademlia node has a 160-bit node *id*. Every message transmits includes its node *id*, permitting the recipient to record the senders existence if necessary. Keys are also 160-bit identifiers. To publish and find $< key, value >$ pairs, Kademlia relies on a notion of distance between two identifiers. Given two 160-bit identifiers, x and y, Kademlia defines the distance between them as their bitwise exclusive or (XOR) interpreted as an integer, $d(x, y) = x \oplus y$.

6. **P-Grid:** Development of P-Grid was motivated by the idea to use a self-organization process (such as in Gnutella or Freenet) to construct an overlay network that is a DHT-like routing infrastructure such that both probabilistic guarantees on search efficiency can be given and resource allocation is optimized. P-Grid is a peer-to-peer lookup system based on a virtual distributed search tree, similar in structure as standard distributed hash tables.

7. **Viceroy [MNR02]:** It is based on the Butterfly [Ref10a] network. Like many other systems, it organizes nodes into a circular identifier space and each node has successor and predecessor pointers. Moreover, nodes are

arranged in $\log N$ levels numbered from 1 to $\log N$. Each node apart from nodes at level 1 have an 'up' pointer and every node apart from the nodes at the last level have 2 'down' pointers. There is one short and one long 'down' pointers. Those three pointers are called the Butterfly pointers. All nodes also have pointers to successors and predecessors pointers on the same level. In that way, each node has a total of 7 outgoing pointers. To lookup an item x, a node n follows its 'up' pointer until it reaches level 1. From there, it starts going down using the 'down' links. In each hop, it should traverse a pointer that does not exceed the target x.

8. **Symphony [MBR03]:** In Symphony, data location is clearly specified by using DHT, but the neighborhood relationship is probabilistically defined. Searching in Symphony is guided by reducing the numerical distance from the querying source to the node that stores the desired data.

2.1.4 Structured Non-DHT-based Systems

There are other structured overlays where a standard graph topology is used instead of applying randomizing hash functions for organizing data items and nodes. System like Mercury organizes nodes into a circular overlay and places data contiguously on this ring. As Mercury does not apply hash functions, data partitioning among nodes is non-uniform.

1. **Mercury [BAS04]:** This system supports multi-attribute range queries with an add-on scheme for load balancing. In this framework, nodes are grouped into virtual hubs each of which is responsible for some query attribute. Nodes inside hubs are arranged in rings in a way similar to Chord. Random sampling is used to estimate the average load on nodes and find light loaded parts.

2. **Skipnet [HJS+03]:** It is an extension of the ring approach that combines the ring structure with the idea of SkipLists [Pug90]. A SkipList is a

sorted linked list that contains supplementary pointers at some nodes that facilitate large jumps in the list in order to reduce the search time of a node in the list. This idea is applied to the ring structure, where nodes maintain supplementary pointers in a circular identifier space. This technique is able to provide the same scalability of traditional DHTs and also supports locality properties. Specifically, SkipNet facilitates placement of Keys based on a nameID scheme that allows the keys to be stored locally or within a confined administrative domain (content locality). In addition, SkipNet also provides path locality by restricting the lookups in the DHT only to domains that may contain the required key. However, since this technique provides (content locality) and (path locality), it trades-off the fault-tolerance of the underlying ring since the content is not uniformly distributed across the ring. Additionally, load balancing issues may also arise in such a system.

3. **Koorde [KK03]:** It is an extension of the Chord protocol and implements deBruijn graphs on top of the ring geometry. A deBruijn graph maintains two pointers to each node in the graph, thereby requiring only constant (i.e. 2 nodes) state per node in the ring. Inheriting the simplicity of Chord, Koorde meets $O(\log n)$ hops per node (where n is the number of nodes in the DHT), and $O(\frac{\log n}{\log \log n})$ hops per lookup request with $O(\log n)$ neighbors per node. Therefore, Koorde is able to improve on the scalability properties of Chord while at the same time maintaining the same amount of state per node. It is worth noting that Koorde's route maintenance protocol is identical to Chord's and therefore it shares the same fault-tolerance capabilities as Chord.

	CAN	Chord	Tapestry	Pastry	Kademlia	Viceroy
Decentralization	DHT functionality on Internet-like scale					
Architecture	Multi-dimensional *id* coordinate space	Uni-directional and Circular Node *id* space	Plaxton-style global mesh network	Plaxton-style global mesh network	XOR metric for distance between points in the key space	Butterfly network with connected ring of predecessor and successor links, data managed by servers
Lookup protocol	Key,value pairs to map a point P in the coordinate space using uniform hash function	Matching Key and Node *id*	Matching suffix in Node *id*	Matching Key and prefix in Node *id*	Matching Key and Node *id* based routing	Routing through levels of tree until a peer is reached with no downlinks
System Parameters	N-number of peers in network and d-number of dimensions	N-number of peers in network and B-base of the chosen peer identifier	N-number of peers in network and B-base of the chosen peer identifier	N-number of peers in network and b-number of bits (B=2b) used for the base of the chosen identifier	N-number of peers in network and b-number of bits (B=2b) of Node *id*	N-number of peers in network
Routing performance	$O(d.N^{\frac{1}{d}})$	$O(log_B N)$	$O(log_B N)$	$O(log_B N) + c$ where c=small constant	$O(logN)$	$O(c.log^2 N)$
Routing State	2d	$logN$	$log_B N$	$Blog_B N + Blog_B N$	$Blog_B N + B$	$logN$
Peers Join/Leave	$2d$	$(logN)^2$	$log_B N$	$log_B N$	$logBN + c$ where c=small constant	$logN$

Table 2.2: Comparison of structured P2P protocols

2.1.5 Hybrid Systems

In recent developments, new generation p2p systems have evolved to combine the advantages of both unstructured and structured p2p networks. This class of systems are referred as hybrid. Structella [CCR04] is one such p2p system that replaces the random graph model of an unstructured overlay with a structured overlay, while still adopting the search and content placement mechanism of unstructured overlays to support complex queries.

1. **Structella [CCR04]:** It is a hybrid proposal based on Pastry. Like the original Gnutella, Structella uses flooding to locate files, but does so in a more efficient way. In particular, Structella uses the underlying structure of Pastry to send no more than one flood message per virtual link. This helps to reduce the flooding cost by a factor of n, where n is the average degree of a node in Pastry.

2. **Kelips [GBL$^+$03]:** It is a $O(1)$-hop overlay that uses an epidemic multicast protocol to exchange overlay membership and other peer states between peers. Kelips divides the peers into k virtual affinity groups with group ids of $[0, k-1]$. To search for an object, the querying peer A in group G hashes the file to the files belonging to group G'. If G' is the same as G, the query is resolved by checking the local intra-group data index. Otherwise, A forwards the query to the topologically closest contact in group G' until the target object is located.

2.1.6 Hierarchical Systems

With increasing pervasive deployments of various p2p overlay architectures, hierarchies are used to connect various nodes in the system. Several important advantages that a hierarchical overlay architecture offers over a flat DHT-based p2p overlay are as follows.

- It reduces the average number of peer hops in a lookup query. Fewer hops per lookup query means that less overhead messages. The higher-level overlay topology consisting of stable superpeers which have more stability. This increased stability allows the lookup query to achieve optimal hop performance.

- It reduces the query latency when the peers in the same group are topologically close. The number of groups is smaller than the total number of peers being query routed and the stability of the higher-level superpeers help to cut down the query delay.

- It facilitates large-scale deployment by providing administrative autonomy and transparency while enabling each participating group to choose its own overlay protocol. Intra group overlay routing is totally transparent to the higher-level hierarchy. If there were any changes to the intra-group routing and lookup query algorithms, the change is transparent to other groups and higher-level hierarchy. That is, any churn events within a group are local to that group in terms of changes and routing tables outside the group are not affected.

A few hierarchical p2p systems available are discussed below along with a brief comparison in Table 2.3.

1. **RChord [LZ06]:** There are three overlays of the RChord: (1) the top layer is the Resource Space Model (RSM) and the p2p Semantic Link Network (p2pSLN); (2) the middle layer is the structured p2p network, that is the Chord overlay; and (3) the bottom layer is the underlying p2p network including various data files and services at each peer. Each peer in R-Chord has three types of neighbors: (1) The neighbors on Chord overlay, which are maintained by the finger table. (2) The neighbors in RSM overlay, which are maintained by the RSM index. (3) The neighbors in p2pSLN overlay, which are maintained by the p2pSLN index.

2. **DR-Chord [SSJKQG07]:** It is based on the Chord overlay structure but divides the single chord ring into two concurrent rings in order to improve both lookup mechanism and reduce the expectation of path length. This two rings are called as clockwise ring and counterclockwise ring. A resource is searched from two directions concurrently. Each node maintains both a clockwise finger table and a counterclockwise finger table each having $m-1$ number of entries. Searching of a data item starts by looking into the clockwise finger table of the querier node n. If it is not present there, then the finger table to which the key belongs to is determined. If it is in the clockwise finger table then that table searched in a way similar to the chord lookup algorithm. If that key is determined to be in the counter-clockwise ring then that ring is searched in a way similar to the chord look up algorithm but the keys are reversed.

3. **Chord^2 [JW07]:** It basically proposes a two-tier architecture. An outer Chord ring is formed with the more stable nodes and the rest of the system nodes remain in the regular Chord ring. The outer ring is called the conduct ring which is given the responsibility of keeping the finger tables of the peers in the regular ring up-to-date. In case of node leaving or joining in the inner ring, some peer in the conduct ring informs the peers in the inner ring which are affected by that event. Thus the maintenance overhead of finger tables is transferred from the regular peers to the conduct ring peers. Therefore, they act as super peers. The selection of more stable nodes to form the conduct ring is due to fact that maintenance overhead of the conduct ring itself must be minimal for this architecture to be effective. Stability of a node may be measured simply on the basis of the time elapsed till its joining or may be a combination of this parameter with some other attributes such as bandwidth capacity, storage, CPU power etc. When a new node n joins the regular chord ring, it builds its finger table in the same way as in normal Chord.

4. **ML-Chord [LHL09]:** It is a multilevel chord protocol. There may be any number of layers based on different parameters under consideration. Also the nodes are divided into two categories: normal peer and BP (bridge peer). Selection of a bridge peer is based on the stability and power of the node. A BP is so called because it connects all the layers together. A layer may contain more than one BP. Also, the BPs are logically connected together in a chord ring. Also, each node may present in more than one layer if it satisfies the criteria for being a particular layer. Each peer contains routing table for each of the layer it belongs to and also stores the BP routing table. The BP routing table is necessary when a data item queried for is not present in a particular layer, then it must be forwarded to the correct layer.

5. **PChord [HLYW05]:** It inherits unchanged Chords successor and finger list to use in PChords routing algorithm and maintenance algorithm. Therefore, the differences between PChord and Chord are the procedures which proximity list takes part in. Proximity list is included into PChord to evaluate the topology of the underlying network. Proximity is weighed by the RTT latency which can be easily got when communications happen between two nodes. An entry in the proximity list contains the IP and identifier of the proximate node. When a new PChord node joins the overlay, it holds an empty proximity list. In such conditions, the routing procedure of this PChord node is the same as common Chord nodes. During the message communication period, this PChord node finds some other PChord nodes near to it with RTT lower than certain value specified beforehand, it adds such kind of nodes to its proximity list. The proximity list increases until the PChord node finds all PChord nodes in the network partition which it belongs to.

   The key modification of routing algorithm in PChord is the choosing of next hop. Next hop is not only decided by the entries in the finger list, but also

decided by the entries in the proximity list. The closest node to the target key in key space will be found out as next hop from local entries of both finger list and proximity list of the current node. Then the entry with that identifier will be chosen as the next hop.

6. **HPS [GEBF+03]:** The general framework for hierarchical p2p system has disjoint clusters. In this system lookup messages are first routed to the destination cluster using a inter-cluster overlay, and then routed to the destination peer through a intra-cluster overlay. Each cluster has one or more superpeers which are selected from certain criteria, e.g. reliability and connectivity. Each normal peer chooses to join one cluster based on the specific requirements of the application. Superpeers form the upper-level overlay. Each cluster can use autonomous intra-cluster overlay lookup service.

7. **HIERAS [XMH03]:** It is a multi-layer DHT that organizes the participating peers into multiple overlay networks (rings) at the different layers. Each of these rings contains a subset of all system peers. Several lower level p2p layers are created, according to the topological proximity metric, besides the upmost level of p2p layer which contains all the participating peers. A routing procedure is executed at the lowest level firstly, and then moves up to a higher level and eventually reaches the upmost level layer. So, the lower the layer of a ring, the smaller is the average link latency between two peers participating in that ring. Thus a lower total routing latency can be achieved.

2.2 Summary

A study on overlay networks is presented in this chapter. A wide range of overlays from centralized p2p to hierarchical p2p are discussed in the survey. It is

	RChord	DRChord	Chord^2	MLChord	PChord	HPS	HIERAS
Architecture	Overlay is categorized into three different types	Based on simple chord but the chord ring is divided into two different ring structure	Proposes two tier architecture	Multilevel chord protocol	Uses proximity list to evaluate topology of the underlying network	Different peers are organized into some disjoint clusters	Multilayer DHT protocol that organize peers into multiple overlay networks
Lookup Protocol	Different routing protocol is carried out on overlays	Routing is done on two different rings based on their existence	First routing is done at the upper level formed by the stbale nodes, then to the next lower level	Routing is carried out based on the routing table on each node in a layer	Next hop is chosen from the proximity list for efficient routing	Each cluster uses autonomous intra-cluster overlay lookup service	Starts at the lower level first, then moves upto the higher level

Table 2.3: Comparison of Hierarchical P2P protocols

found that p2p overlays are increasingly becoming popular choice for most of the Internet based applications. Among these the hierarchical overlays are the most recent trend in p2p area. The heterogeneity of nodes leads one to treat the nodes of p2p systems in many different ways. it is observed from the survey that some of the main challenges that p2p systems should face are - scalability, performance, adaptability, load balancing, resilience, and providing security and incentive mechanism.

Chapter 3

SHP: A Structured Hierarchical Overlay Protocol

The structured overlays use a specific routing geometry and also support key-based routing in a way that object identifiers are mapped to the peer identifier address space and an object request is routed to the nearest peer in the peer address space. Peer-to-peer systems using key-based routing are also known as distributed object location and routing (DOLR) systems. The distributed hash table (DHT) is a specific type of DOLR.

The structured overlays are designed with many dimensions. The design of structured overlay systems are based on maximum number of hops taken by a request on an overlay of n peers, organization of the peer address space, next-hop decision criteria, geometry of the overlay, overlay maintenance and locality and topology awareness. The identifier space is large in a p2p system. The address space may be flat or hierarchical. Hierarchical overlay consists of n-tier overlays whereby the nodes are organized into disjoint groups. The overlay routing to the target group is done through an intergroup overlay; then intra-group overlay is used to route to the target node. The hierarchical overlay architecture of the Internet offers several important advantages over the flat DHT-based p2p overlay

such as reduces the average number of peer hops in a lookup query and minimizes the query latency. The hierarchical overlay facilitates large-scale deployment by providing administrative autonomy and transparency while enabling each participating group to choose its own overlay protocol. Intra group overlay routing is totally transparent to the higher-level hierarchy. If there were any changes to the intra-group routing and lookup algorithms, the change is transparent to other groups and higher-level hierarchy.

In a p2p system large number of autonomous systems participate and form self organizing networks, which are based on the overlay networks i.e. over the top of conventional IP with no centralized structure. The distributed object location service is used to establish communication among nodes, which in turn requires efficient routing for better performance of the system as a whole. The participants of a p2p systems are typically individuals and organizations without an out-of-band trust relationship and systems are heterogeneous in nature.

The aim of this chapter is to develop a *Structured Hierarchical overlay Protocol* (SHP) to improve the overall performance of the system. The overlay that is proposed should be able to handle both point and range queries and also should be capable of handling transient node population in the system with inherent hierarchical property. One of the major applications of p2p systems is content distribution. Millions of nodes may join a p2p network to share resources spread over the world. It is detrimental to the performance of the system if all the nodes are considered identical. Nodes in a network may differ in terms of available bandwidth, processing power, stability, privacy & security, reliability, amount of data shared and storage capacity.

The nodes in the system are classified as given in section 3.1 and the system architecture is discussed the system architecture in section 3.2. The process of

joining and leaving of nodes and query processing are discussed in sections 3.3 and 3.4, respectively. A load balancing mechanism is proposed in section 3.5. The storage requirement of the system is discussed in section 3.6. The section 3.7 provides experimental results and related works for this chapter are presented in section 3.8. Finally, the summarization of this chapter is done in section 3.9.

3.1 Node Classification

It has been observed that all the nodes available in a p2p system are not homogeneous. An analysis [AH00] has shown that 70% of the Gnutella users share no files and 90% of users do not response to the query. Another Gnutella analysis [ZSR06] shows that 7% peers share more files than all other peers and 47% queries are responded by the top 1% peers. Definitely, *temporary nodes* have influenced the analysis due to their short span of life in p2p systems. For example, mobile devices have limited bandwidth and power, variable capacity and asymmetric links. These links are error-prone and hence network congestions are more, so it is difficult to achieve guaranteed quality of service (QoS). Eventually the presence of temporary nodes can not be expected in p2p networks for long time. To give importance to nodes according to their capabilities, nodes are taxonomized into three categories as follows.

i. Temporary nodes

ii. Stable nodes or peers and

iii. Fully-stable nodes or superpeers

1. **Temporary nodes**: A newly joined node to the network is called a temporary node. It does not have own routing table and hence no corresponding entries in the DHT. These nodes connect at the last level of hierarchy and

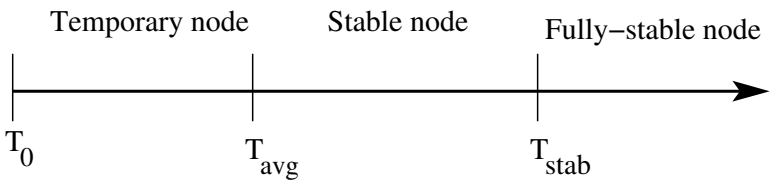

Figure 3.1: Categorization of nodes.

Start	Successor
$0+1$	5
$0+2$	5
$0+4$	5
$0+8$	10
$0+16$	20

Table 3.1: PRT of node $id = 0$ as shown in Figure 3.3

may be disconnected after accessing resources from p2p network. A temporary node knows the *id* of the contacted node and access information from the p2p network.

2. **Stable node**: A temporary node becomes a stable node or peer after satisfying stability period T_{avg} as shown in Figure 3.1. T_{avg} is the observed average life-time of a node in the system. Since this parameter can not be calculated beforehand so it is derived empirically at the time of instantiation of the network. Stable nodes have their own routing table known as *Peer Routing Table* (PRT). The structure of PRT is same as finger table used in Chord. Table 3.1 shows the PRT for a node (node $id = 0$) of the Group as shown in Figure 3.3. A stable node has information about its group, its successor and predecessor.

3. **Fully-stable node**: A stable node gets promoted to fully-stable node or superpeer after completing the duration of T_{stab} and satisfying some other constraints like processing power, storage capacity, etc. T_{stab} is a empirically set initial parameter which determines the stability period of a fully-stable

Group	Key interval	Fully-stable node vector
G_0	$id_a - id_b$	F_0
G_1	$id_c - id_d$	F_1
G_2	$id_e - id_f$	F_2
G_3	$id_g - id_h$	F_3

Table 3.2: Group Routing table. Here $[id_a - id_b]$ denotes the *key* interval values in the group G_0 and so on.

node. The value of T_{stab} is chosen sufficiently higher than T_{avg} so that network will not become a complete mesh. Hence, this can be used to control the number of fully-stable nodes in the system. These nodes are assumed to have ideal characteristics, higher availabilities and predicted to be available in future for long duration compared to other nodes. A fully-stable node contains PRT and Group Routing Table (GRT). GRT, as shown in Table 3.2, stores the addresses of all fully-stable nodes of all other groups in a network. Therefore, inter-group lookup is reduced to $O(1)$. Fully-stable nodes act as gateways for inter-group communication.

3.2 System Architecture

Figure 3.2 shows the high level architecture of SHP where nodes are organized into groups. Number of nodes in a group may vary from one group to another. For example, some groups may have thousands of nodes while others may have only hundred. Each group maintains a ring structure using the well-known successor-maintenance algorithm [CLGS04] like Chord. Within a group, nodes are relatively close to each other and hence intra-group latency is sufficiently reduced. At the top level of the hierarchy, joining and leaving of nodes is rarely possible, because fully-stable nodes are relatively stable and expected to be available for long time.

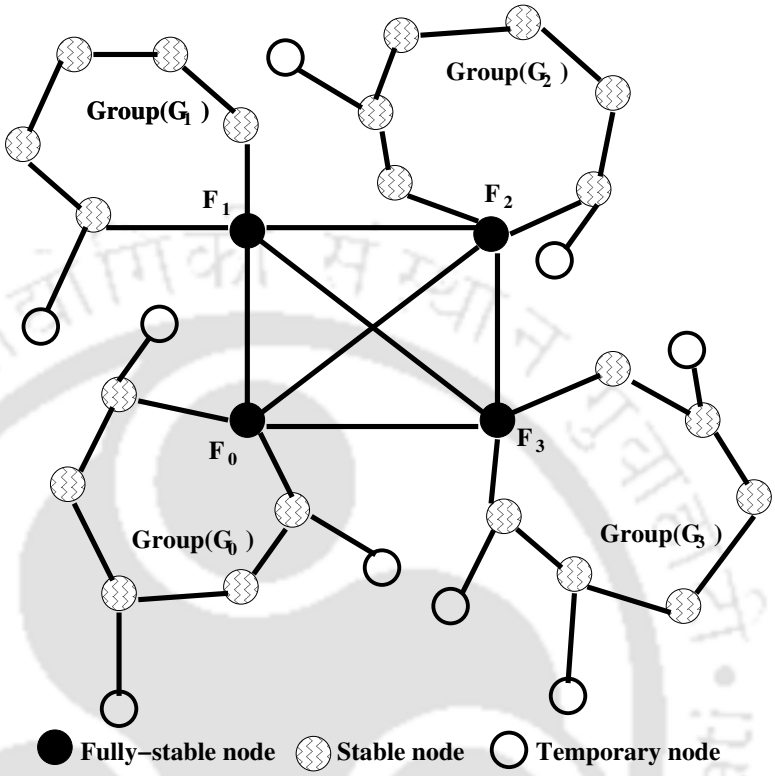

Figure 3.2: Illustrating High-level architecture of SHP

As it is evident from the Figure 3.2, only stable nodes are part of the Chord ring structure (which is considered to be the second level of the hierarchy) in a group. Temporary nodes are in the last level of the hierarchy. The first level of the hierarchy maintains a mesh structure which consists of one of the fully-stable node from each Chord ring. Temporary nodes may join or leave the network at their will without effecting the performance of the system.

3.2.1 Bootstrapping

It is assumed that first g nodes that join the p2p network trivially qualify for fully-stable nodes, where g is the initial number of groups. Fully-stable nodes maintain the time-stamp of newly joined nodes and form a group. Number of fully-stable nodes in a group may increase further to G_{max}, where G_{max} is the

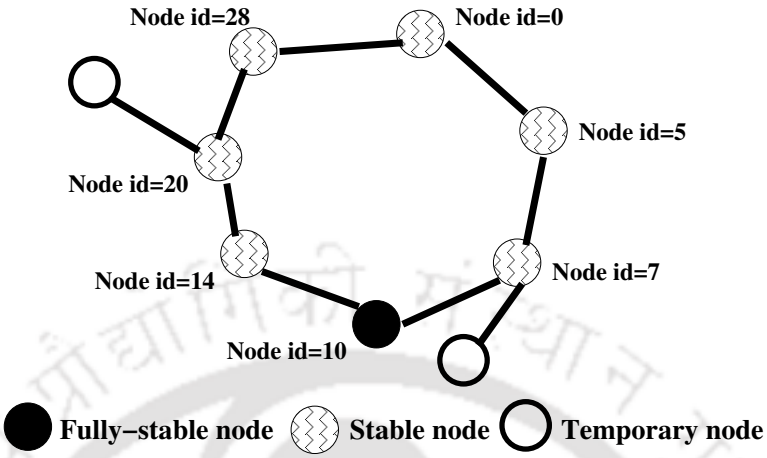

Figure 3.3: A Group with 5-bit node *id*.

maximum number of nodes in a group. When number of nodes in a group becomes more than G_{max}, the group is divided into two groups choosing one of the stable nodes as fully-stable node.

3.3 Joining and Leaving of Nodes

Any node that wants to join a p2p network, joins as a temporary node in the last level of hierarchy. To reduce the intra-group lookup, a node first sends k ping messages to different landmarks and finds the average Round Trip Time (RTT) [RHKS02]. It contacts the group which is closest to it based on the RTT. As the node joins as a temporary node, it does not have any routing table upto T_{avg} time units. When a temporary node crosses the T_{avg} time duration, it is expected that it has high probability to remain active in the network. So, data shared by a stable node is available in the p2p system with high probability. Hence, joining of a stable node can smoothly be handled by the following three steps.

- Initialization of PRT of newly upgraded stable node.

- Update PRTs of existing stable nodes.

- Transfer of *keys* from its successor.

Leaving of a stable node can be handled by the following two steps.

- Transfer of *keys* from leaving node to its successor node.

- Update PRTs of existing stable nodes.

3.3.1 Effect of Temporary Node

Since a temporary node maintains no routing table, it may easily leave the p2p network without informing any other nodes.

Let n be the number of nodes in the system. Hence, $\frac{n}{g}$ is the average number of nodes in a group where g is the number of groups in the system. Let λ_j and λ_l (where $\lambda_j \geq \lambda_l$ and $\lambda_j, \lambda_l \geq 0$) be the rate of joining and leaving of temporary nodes, respectively. Let M_{ccp} and M_{shp} be the number of updates required in time span T_{avg} to repair the routing table in conventional Chord protocol and SHP, respectively.

Chord handles the joining or leaving event of nodes with no more than $O(\log n)^2$ updates with high probability. Hence, it is assumed that $(\log n)^2$ updates are required to handle joining/leaving operation. Therefore,

$$\begin{aligned} M_{ccp} &= (\log n)^2 \times \lambda_j \times T_{avg} + (\log n)^2 \times \lambda_l \times T_{avg} \\ &= (\lambda_j + \lambda_l) \times (\log n)^2 \times T_{avg} \end{aligned}$$

In SHP, temporary nodes do not have entries in routing table. Since nodes which are available beyond T_{avg} time affect the routing tables of existing stable nodes and fully-stable nodes, the number of updates required to repair the routing table in time span T_{avg} is given by

$$M_{shp} = (\lambda_j - \lambda_l) \times (\log(\frac{n}{g} - \frac{(\lambda_j - \lambda_l) \times T_{avg}}{g}))^2 \times T_{avg}$$

If the rate of leaving nodes is too frequent i.e. $\lambda_j \approx \lambda_l$, SHP saves $2\lambda_j \times (\log n)^2 \times T_{avg}$ updates in time span T_{avg}.

Now considering the average case scenario as leaving of half of the nodes from the system in T_{avg} i.e. $\lambda_l = \frac{\lambda_j}{2}$. Then

$$\frac{M_{shp}}{M_{ccp}} = \frac{\frac{\lambda_j}{2} \times (\log(\frac{n}{g} - \frac{\frac{\lambda_j}{2} \times T_{avg}}{g}))^2 \times T_{avg}}{\frac{3\lambda_j}{2} \times (\log n)^2 \times T_{avg}}$$

$$= \frac{(\log(\frac{n}{g} - \frac{\lambda_j \times T_{avg}}{2g}))^2}{3 \times (\log n)^2}$$

Where $g \geq 1$ and $T_{avg} \geq 0$

Hence $\frac{M_{shp}}{M_{ccp}} \leq \frac{1}{3}$

Therefore, SHP saves at least 66.67% updates in average case to repair the routing table due to frequent joining and leaving of temporary nodes.

3.3.2 Effect of Stable Node

The joining/leaving of a stable node in a group needs $O((\log(n_s + n_f))^2)$ updates to repair the routing table in contrast to $O((\log(n_t + n_s + n_f))^2)$ in Chord protocol, where n_t, n_s and n_f are the number of temporary, stable and fully-stable nodes in a group, respectively.

3.3.3 Effect of Fully-stable Node

The joining of a fully-stable node needs update of GRT. Thus, number of updates require during joining of a fully-stable node is $O(N_f)$, where N_f is the total number of fully-stable nodes in the system. The assumption is that leaving of a fully-stable node is very rare as this node is most powerful and highly stable.

Even if a fully-stable node leaves the system, the cost to repair the routing table is $O((\log(n_s + n_f))^2)$ and $O(N_f)$ for PRT and GRT, respectively.

3.4 Query Processing

The SHP supports point query and range query. A single data item is searched in point query, where the key is hashed to find the identifier for the file. The node responsible for that file is located and the file is retrieved. A data item may be found within the group (intra-group) or outside the group (inter-group). A range of data items are searched in range query, where corresponding node is found for each data item and retrieved.

3.4.1 Range Query

DHT-based system like Chord provides an efficient solution for point queries. But uniform distribution of hash function disturbs the data relationship and does not have support for range query.

To handle range queries in SHP, complete data space is divided into some regions. Each group is associated with a part of the data space. At the time of startup number of regions in the data space is equal to the number of groups. A fully-stable node holds information about the group and hence it knows the part of data space corresponding to particular group.

Let the complete key space be $[AAA - DDD]$ with three groups in the system. The division of key space to groups is $[AAA - BBB)$, $[BBB - CCC)$, $[CCC - DDD)$. The format of the Group Routing Table is shown in Table 3.3.

Internal to the group, key space is divided into intervals and each interval is assigned to one node. A node is allowed to hold more than one interval but

Group	Key interval	Fully-stable node vector
G_1	$AAA - BBB$	F_1
G_2	$BBB - CCC$	F_2
G_3	$CCC - DDD$	F_3

Table 3.3: A GRT showing key space $AAA - DDD$

an interval is never mapped to more than one node. Each node maintains the following information apart from the routing table which is used for routing.

- *Predecessor Range Node*: A node which contains the immediate previous key range is called as predecessor range node. For instance let A, B, C be three nodes which hold key lying in the range $[AAA{-}BBB)$, $[BBB{-}CCC)$, $[CCC - DDD)$ respectively. Then A is called the predecessor range node of B.
- *Successor Range Node*: A node which contains the immediate next key range is called as successor range node. In the above example, C is the successor range node of B.
- *Key Interval*: The key space can be arranged in ordered manner and divided into several intervals. Each interval is known as key interval.

Table 3.4 shows the range table of node B as stated above.

Key interval	Predecessor Range node	Successor Range node
$[BBB - CCC)$	A	C

Table 3.4: Range Table of node B

Range Query Processing

Let the range of data maintained by a group be X_1 to X_2. Let the query range be from p to q. When a node gets a query, it checks whether the query falls

in the region maintained by the group. If the query does not fit in the interval maintained by the group, then it is forwarded to one of the fully-stable nodes of that group which in turn forwards it to the respective group, where a desired data space is available. The following steps are used to process a range query.

i. If (p to q) falls in the range (X_1 to X_2) then the node runs the regular Chord algorithm to find the node holding data p. Then the query is forwarded to the successor range node till the range of data is obtained.

ii. If ($p < X_1$) and q falls in the range of the data maintained by the group, then the range $[p - X_1)$ is forwarded to one of the fully-stable nodes for further processing. Step i is used for the range $[X_1 - q]$.

iii. If p falls in between the range and q is out of the data range maintained by the group, then for the range $[p - X_2)$ step i is executed. For the range $[X_2 - q]$ the message is sent to the fully-stable node of the group for locating the desired group.

The message format for range query is as follows.

$Query(hash(Lkey), k_1, k_2, NodeID(id))$, where $Lkey$ is the starting point of the interval containing the desired data range i.e. $LKey = k_1 - (k_1 \% j)$; k_1 is lower bound of range of data; k_2 is upper bound of range of data; id is identifier of the source node and j is key interval size.

Assuming uniform distribution of nodes to all groups, lookup complexity of range query is given by $O(\log(\frac{n}{g}) + \frac{R}{j})$, where g is the number of groups; n is the total number of nodes and R is the length of range query. The lookup complexity can be simplified as $O(\log(\frac{n}{g}))$.

3.4.2 Point Query

Suppose a node P_i of group G_i makes a query for key k. The fully-stable node F_i finds the responsible group for key k. If k is found within G_i, then it returns with

complexity $O(\log n_i)$, where n_i is the number of nodes in group G_i. Otherwise F_i sends request to the fully-stable node where the key k belongs. This takes $O(1)$ lookup step, since every fully-stable node knows about fully-stable nodes of other groups. Suppose F_l of group G_l recieves the request, then key k is retrieved with complexity $O(\log n_l)$, where n_l is the number of nodes in group G_l. Therefore, it is found that effective complexity to execute a point query is $O(\log \frac{n}{g})$.

The message format for point query is shown below.

$Query(hash(Lkey), k, NULL, NodeID(id))$, where $Lkey$ is the starting point of the interval containing the desired data range i.e. $Lkey = k - (k\%j)$; k is the key of data object; id is identifier of the source node; j is key interval size.

The first field generates the key id for key interval in which key k lies. The second field specifies exact key within that key interval. NULL in the third field indicates that it is a point query. Finally, the last field specifies the node id of the *querier node* (the node which makes query).

For example, let the key intervals be [0-10), [10-20), [20-30), ... and a querier node makes query for the key 15. Then $j = 10$ and $Lkey = 15$ - (15 % 10) = 10. Hence, $hash(10)$ gives the key id for key interval [10-20) within which key 15 lies.

3.5 Load Balancing

Nodes are placed at three different levels of hierarchy in the system. Fully-stable nodes in the first level of hierarchy are most trustworthy and are expected to be in the network for most of the time. Data maintained by the group of nodes determines the load on the group. Some part of the data space might be queried a lot compared to that of other part. It is impractical to assume uniform distribution of queries in the data space. Hence, the system is not balanced and load balancing is needed for stable nodes and fully-stable nodes.

3.5.1 Grading of Nodes

Each fully-stable node periodically collects the load of stable nodes in its group and grades them based on the following definition of load. The *Load of a node*(L_n) is defined as the ratio of amount of data requests it receives to the amount of data it can serve.

i. Lightly loaded nodes: $L_n \leq 0.5$.

ii. Normal loaded nodes: $0.5 < L_n \leq 1$

iii. Heavily loaded nodes: $L_n > 1$.

The main objective of load balancing is to ensure that the load of each node remains between 0 and 1 i.e. $0 \leq L_n \leq 1$.

3.5.2 Cost of Load Balancing

The following steps are required to evaluate the cost of load balancing in SHP overlay system.

Step-1. When a node P gets overloaded, it reduces the data space by transferring some load to either successor range or predecessor range nodes. Moving load to the successor range or predecessor range nodes does not incur any message cost.

Step-2. If both the successor range and predecessor range nodes are overloaded and cannot handle any more load, then the node searches for a node in the proximity and transfers some load to it.

Step-3. If the node P can not find any lightly loaded nodes, then it sends a request message "NEED-NODE" to the fully-stable node of the group. The fully-stable node checks for a lightly loaded node in the groups. If a lightly loaded node is found in some other group, then it is deleted from that group and

added to the group to which P belongs. Then P can transfer some of its load to this lightly loaded node.

Hence, load balancing involves addition and deletion of node in the SHP overlay system. Therefore, if $\frac{n}{g}$ is the average number of nodes in a group, then the cost of load balancing is $O(\log \frac{n}{g})^2$.

3.6 Storage Requirement

Each fully-stable node maintains two routing tables, Group Routing Table (GRT) and Peer Routing Table (PRT) for inter-group lookup and intra-group lookup, respectively. Stable nodes maintain only PRT, while no routing table is maintained by temporary nodes.

Since these two routing tables are different, records of GRT differ from the size of records in PRT.

Let B_g = Size of each record in GRT

B_p = Size of each record in PRT

N_t = Number of temporary nodes in p2p network

N_s = Number of stable nodes in p2p network

N_f = Number of fully-stable nodes in p2p network

n = Total number of nodes in p2p network

g = Number of groups.

Since number of temporary nodes in p2p networks depends on the joining and leaving rate,

$$N_t = ((\lambda_j - \lambda_l) \times T_{avg})$$

Assuming that each group G_i contains $\frac{n}{g}$ nodes, $0 \leq i \leq (g-1)$, size of the routing table at fully-stable node is the sum of size of GRT and PRT stored on

it.

Let SR_{shp} and TSR_{shp} are the node storage requirement and total storage requirement in SHP, respectively and SR_{ccp} and TSR_{ccp} are node storage requirement and total storage requirement, respectively in conventional Chord protocol.

$$SR_{shp}(F_i) = B_g \times g + B_p \times \log(\tfrac{n}{g}) \text{ ——- (1)}$$

Since stable node only contains PRT,

$$SR_{shp}(S_i) = B_p \times \log(\tfrac{n}{g}) \text{ —— (2)}$$

The storage requirement for all fully-stable nodes and stable nodes are obtained as follows.

From equation (1),

$$\begin{aligned} &\sum_{i=0}^{N_f-1} SR_{shp}(F_i) \\ &= N_f[B_g \times g + B_p \times \log(\frac{n}{g})] - - - -(3) \end{aligned}$$

From equation (2),

$$\begin{aligned} &\sum_{i=0}^{N_s-1} SR_{shp}(S_i) \\ &= N_s \left[B_p \times \log\left(\frac{n}{g}\right)\right] \\ &= (n - N_f - (\lambda_j - \lambda_l) \times T_{avg}) \times \\ &\quad \left[B_p \times \log\left(\frac{n}{g}\right)\right] - - - -(4) \end{aligned}$$

Total storage requirement for routing table is given from equations (3) and (4)

$$TSR_{shp} = N_f[B_g \times g + B_p \times \log(\tfrac{n}{g})] + N_s[B_p \times \log(\tfrac{n}{g})]$$

$$= N_f \times B_g \times g + N_f \times B_p \times \log(\tfrac{n}{g}) + (n - N_f - (\lambda_j - \lambda_l) \times T_{avg}) \times [B_p \times \log(\tfrac{n}{g})]$$

$$= N_f \times B_g \times g + (n - (\lambda_j - \lambda_l) \times T_{avg}) \times [B_p \times \log(\tfrac{n}{g})] \text{ —- (5)}$$

From equation (5), let us approximate TSR_{shp} as

$$TSR_{shp} = N_f \times B_g \times g + n \times B_p \times \log \tfrac{n}{g}$$

In conventional Chord protocol PRT size,

$$SR_{ccp} = \log n \times B_p$$

Hence, $TSR_{ccp} = n \times \log n \times B_p$

In SHP the number of tables stored on a node depends on the node type. For example, temporary nodes, stable nodes and fully-stable nodes store 0, 1 and 2 tables respectively, while the conventional Chord protocol stores single table at each peer. Therefore, TSR_{shp} and TSR_{ccp} are considered for comparing the storage requirement of SHP and CCP.

Let $TSR_{shp} = TSR_{ccp}$

$$N_f \times B_g \times g + n \times B_p \times \log \tfrac{n}{g} = n \times \log n \times B_p$$

$$N_f \times B_g \times g + n \times B_p \times \log n - n \times B_p \times \log g = n \times \log n \times B_p$$

$$N_f \times B_g \times g - n \times B_p \times \log g = 0$$

Assuming that record size of PRT is same as record size of GRT i.e $B_p = B_g$. Then

$N_f = \frac{n \times \log g}{g}$

S.No	Conditions	Remark
i.	$(N_f < \frac{n \times \log g}{g}) \& (B_p = B_g)$	$TSR_{shp} < TSR_{ccp}$
ii.	$(N_f = \frac{n \times \log g}{g}) \& (B_p = B_g)$	$TSR_{shp} = TSR_{ccp}$
iii.	$(N_f > \frac{n \times \log g}{g}) \& (B_p = B_g)$	$TSR_{shp} > TSR_{ccp}$

Table 3.5: Comparison of Storage Requirement

It is observed from the Table 3.5 that satisfying condition (*iii*) is rare because number of fully-stable nodes are limited as mentioned in section 3.3.3. Hence, SHP requires less overall storage space compared to conventional Chord protocol.

3.7 Experimental Results

In this section, first we have given a brief outline of a simulator and its importance. Then we have shown the observations of experiments along with discussions.

3.7.1 Simulator

Any scientific community including the computer science community requires that the results of the models developed for specific purposes are tested so that reproducible results are provided. To achieve this, analytical solutions, simulations and experiments are the methods that are used. However, in the area of p2p, they give rise to a number of challenges.

The Analytical approach is one where a mathematical model of the system is examined. It can provide an analytical solution, but only if the model is simple. While this approach is being used in p2p research, as it requires a simple model,

many of the complexities of real-world p2p systems have to be ignored.

In cases where applying analytical methods prove too complex, an alternative is to run experiments with the actual system. Peer-to-peer systems can consist of a large number of nodes and any experiment on even a relatively small scale of a few thousand nodes would be impractical, or even impossible. Even it may be feasible to acquire the resources necessary to run such an experiment; there are other impracticalities that must be considered. Applying changes to an experiment such as modification of the protocol running at each node and the topology of the network would be difficult and certainly time consuming if the experiment were performed on a large scale.

Simulation techniques are used in most cases which provide a flexible framework to test the developed applications. Even though simulators have some disadvantages, they do address the impracticalities of mathematical analysis and experimentation. However simulations are not seen as strictly independent from the analytical approach and experimentation. If possible analytical approaches should be utilized and proved using a simulator, similarly simulation results should if possible be validated by experiments with the actual system.

S. Naicken et al. [NBLR06] discuss about DHTSim, P2PSim, Overlay Weaver, PlanetSim, PeerSim, GPS and Neurogrid. Most of these simulators are developed out of the necessity to fulfill a particular requirement. In general none of the simulators give a common platform to design and test any p2p structure. In particular, none of the available simulator provides any classification of nodes considering the heterogeneity that is inevitable in any computer network. Therefore it is difficult to implement the SHP model using these simulators which is based on the taxonomy of nodes. Therefore, it is decided to develop a new simulator (it is named as *shpsim*) to simulate SHP and Chord together and a relative comparison

is done between two by observing the simulation results. The detailed design and development procedures are included in Appendix-*A*. A comparison of *shpsim* with some existing simulators is given in Table 3.6.

	Oversim	OpenChord	PlanetSim	*shpsim*
GUI	Yes	No	Yes	Yes
Source Code	Open	Open	Open	Open
Network supported	Many	One	Few	Two
Finger Table	Viewable	Viewable	Viewable	Viewable
Real time simulation	Yes	No	No	No
Manual events possible	No	Yes	Yes	No

Table 3.6: Comparison of Simulators

An *event file* is generated as input to the simulator. An event file consists of records for different events with the following fields.

Time: The time at which the event is generated.

Event: Nature of event i.e. joining /leaving of a node, searching of item(s) using point query and range query.

NodeID: Identifier of the node which has generated the event.

A few sample record in the event file is shown in Figure 3.4.

3.7.2 Joining and Leaving of Nodes

When a node wants to join the system, it first contacts any fully-stable or stable node and joins as a temporary node. After time-stamp T_{avg} it executes the JOIN algorithm of Chord and joins the group at its specific location (as determined by the hash function). The joining and leaving process is implemented as in Chord, considering the group as a Chord ring. Since the number of nodes in a group is less than the total number of nodes in the system, the number of messages required for joining and leaving is less than Chord.

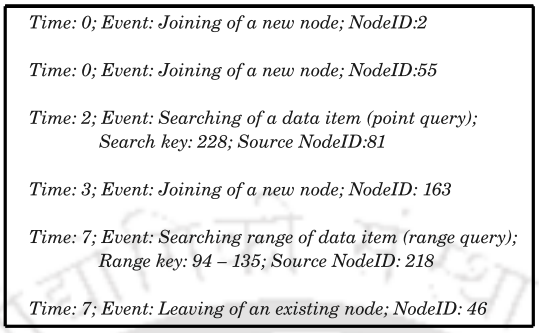

Figure 3.4: A snapshot of records in the event file

Figure 3.5 shows the comparison of messages required for joining of nodes in Chord and SHP for various sizes of network. For our experiments, number of nodes taken as 256, 512, 1024 and 2048. It is seen from the Figure 3.5 that a node in SHP takes less number of messages to join in the system in comparison to Chord and the same has been shown for different sizes of network.

Figure 3.6 shows effect of joining of nodes with different group sizes in SHP. The number of nodes in the system is taken as 2048 and number of groups cosidered as 8, 16, 32 and 64. It is found form Figure 3.6 that performance of SHP is better as the number of nodes in a group is more than the total number of groups in the system. But the efficiency of SHP is reducing when the total number of groups in the system is nearing or greater than the number of nodes in a group, even smaller than the Chord as shown in Figure 3.6-[C] and Figure 3.6-[D].

Figure 3.7 represents joining of nodes with varying T_{avg} value keeping size of network and T_{stab} value fixed. As the T_{avg} value increases the number of temporary nodes in the system increases, so messages required to join a node in SHP decreases as depicted in Figure 3.7.

Figure 3.8 represents joining of nodes with varying T_{stab} value keeping size of network and T_{avg} value fixed. As the T_{stab} value approaches T_{avg} value, the nodes in the system frequently upgrade to fully-stable node, which need to build Group Routing Table (GRT) and Peer Routing Table (PRT). It is observed that in all cases SHP takes less messages to join nodes as compared to Chord as shown in Figure 3.8-[A], Figure 3.8-[B], Figure 3.8-[C] and Figure 3.8-[D].

Graphs shown in Figure 3.9, 3.10, 3.11 and 3.12 indicate less updates required for leaving of nodes in SHP and Chord. Since leaving of a temporary node does not effect the SHP overlay, hence it is seen that for some initial values leaving of nodes take zero message pass. Therefore, the leaving of nodes in SHP takes less messages in comparison to Chord which supports theoretical assumptions.

3.7.3 Query Processing

In SHP more than one key interval can be mapped to a node but a key interval cannot be partially mapped to more than one node. It is observed that number of hops required for point and range queries is less for SHP in comparison to Chord. In SHP, if a data item is found within the same group then it is found within $O(\log \frac{n}{g})$ steps; otherwise, it needs another $O(\log \frac{n}{g})$ steps. For range query difference in number of messages required is more since Chord doesn't support range query (i.e. each data item within a range is retrieved using individual point query), whereas in SHP it executes the query with first key of the range as discussed in section 3.4.1.

Figure 3.13, 3.14, 3.15 and 3.16 illustrate the number of hops required to execute point query in different conditions as discussed in subsection 3.7.2. It is observed that in all situation execution of point query in SHP has improvement

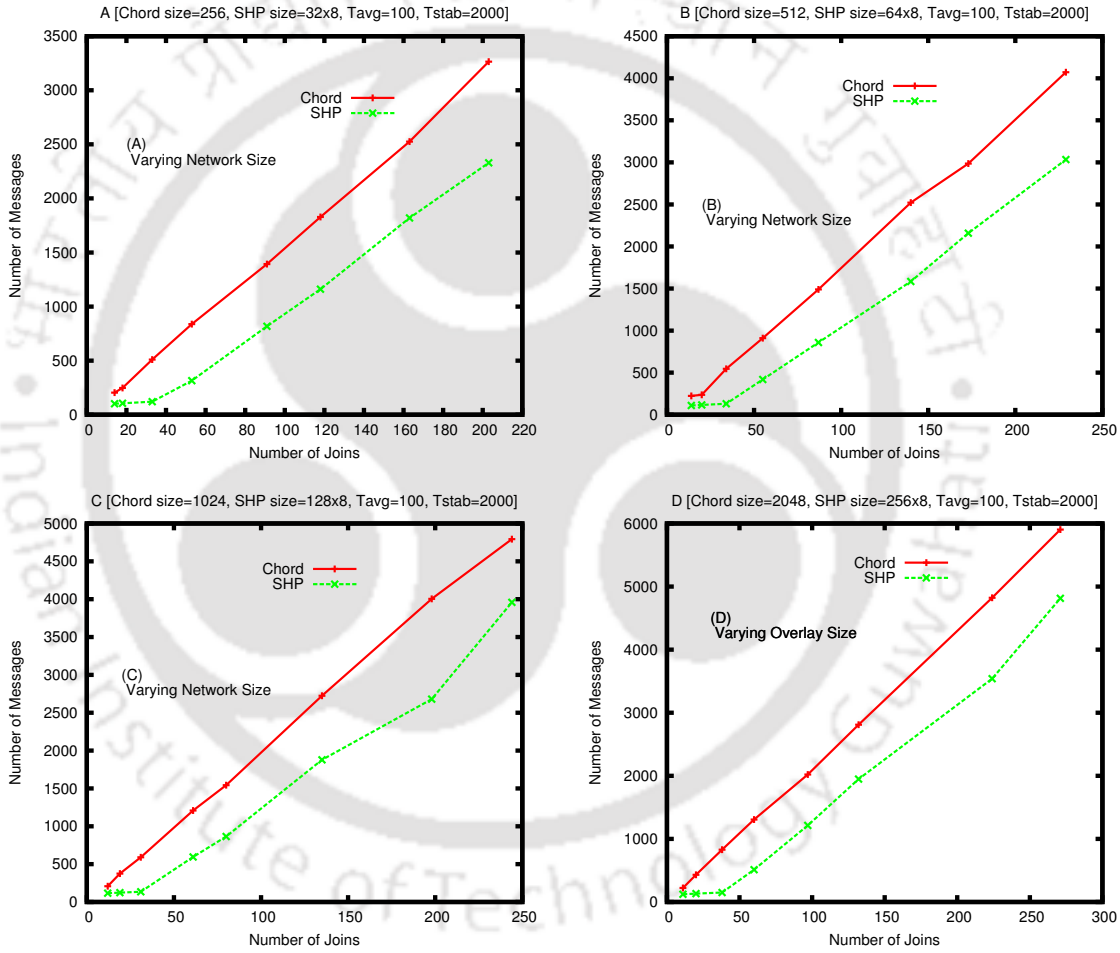

Figure 3.5: Graph showing number of messages required to join nodes in Chord and SHP with different network size

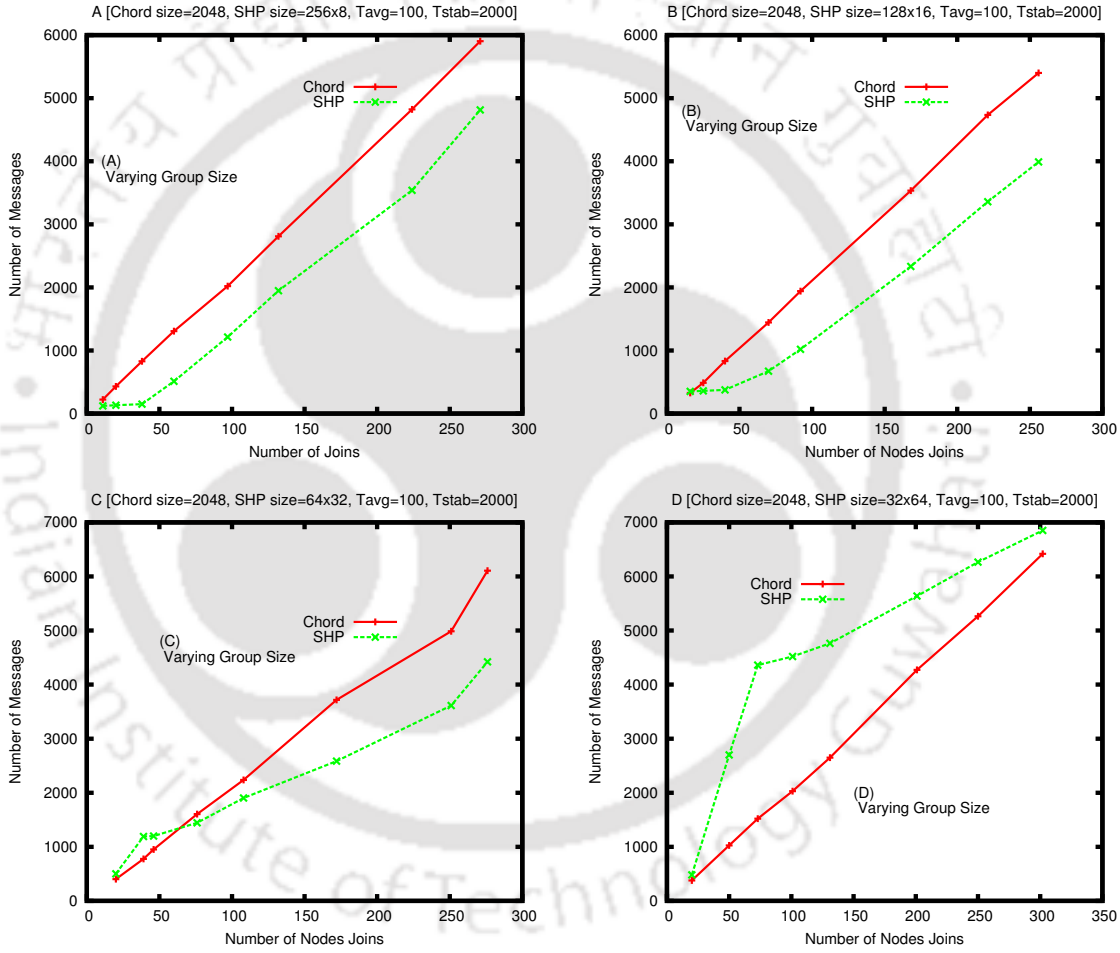

Figure 3.6: Graph showing number of messages required to join nodes in Chord and SHP with different group size of same network size

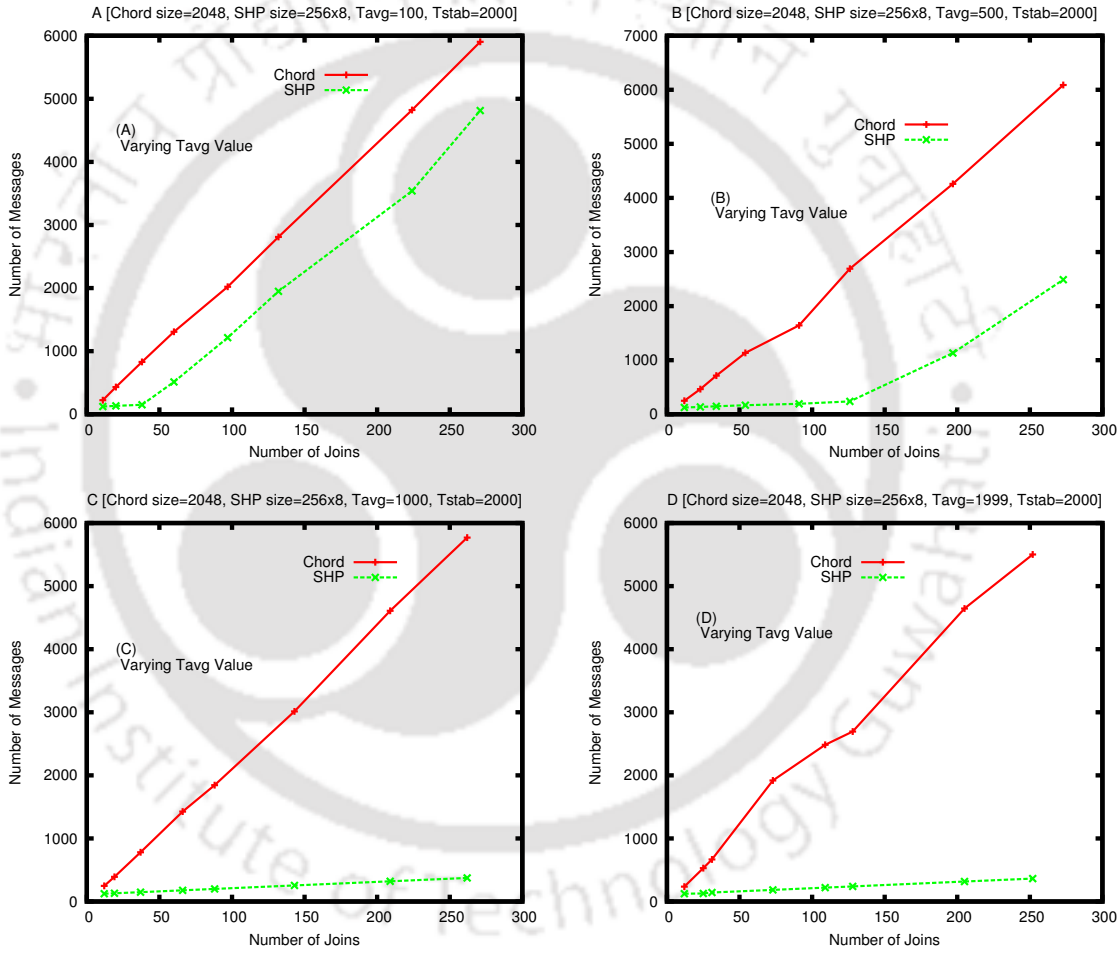

Figure 3.7: Graph showing number of messages required to join nodes in Chord and SHP with different T_{avg} value of same network size

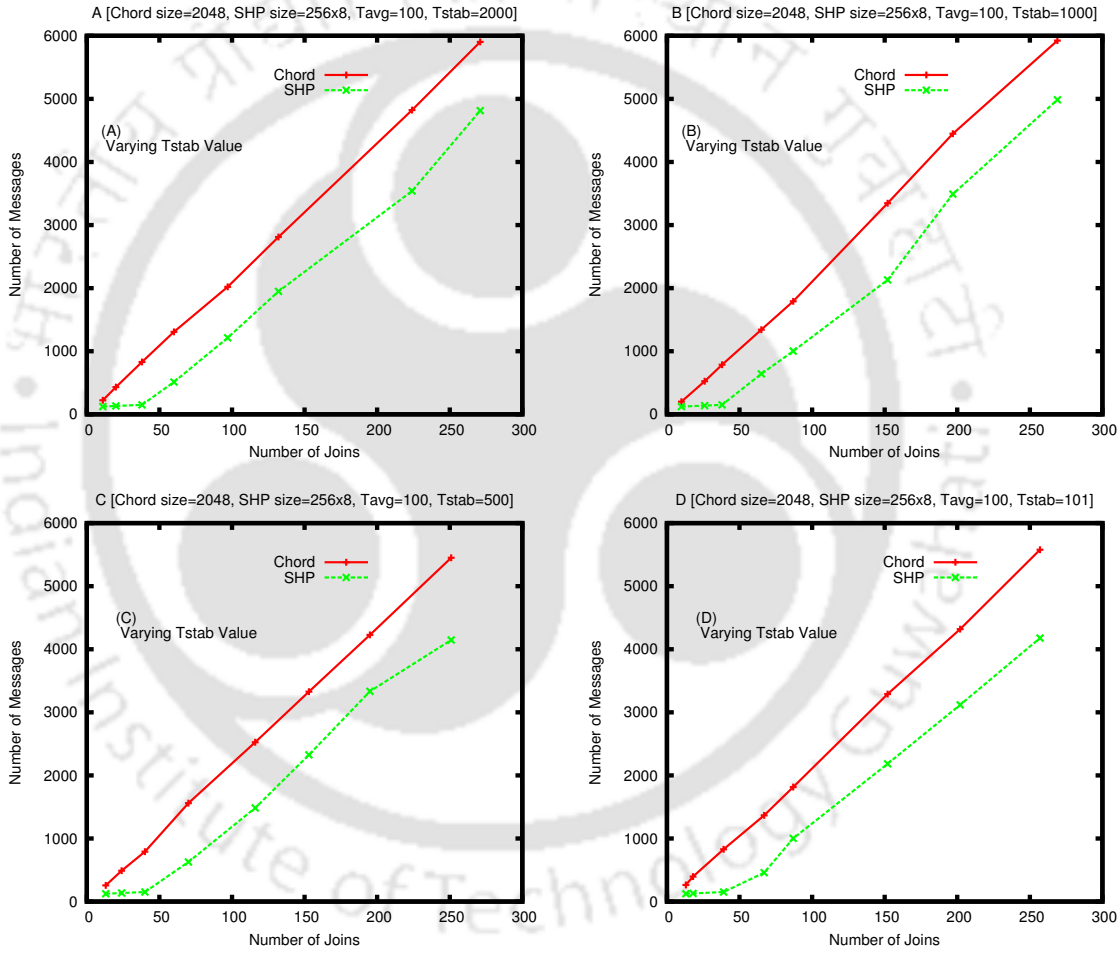

Figure 3.8: Graph showing number of messages required to join nodes in Chord and SHP with different T_{stab} value of same network size

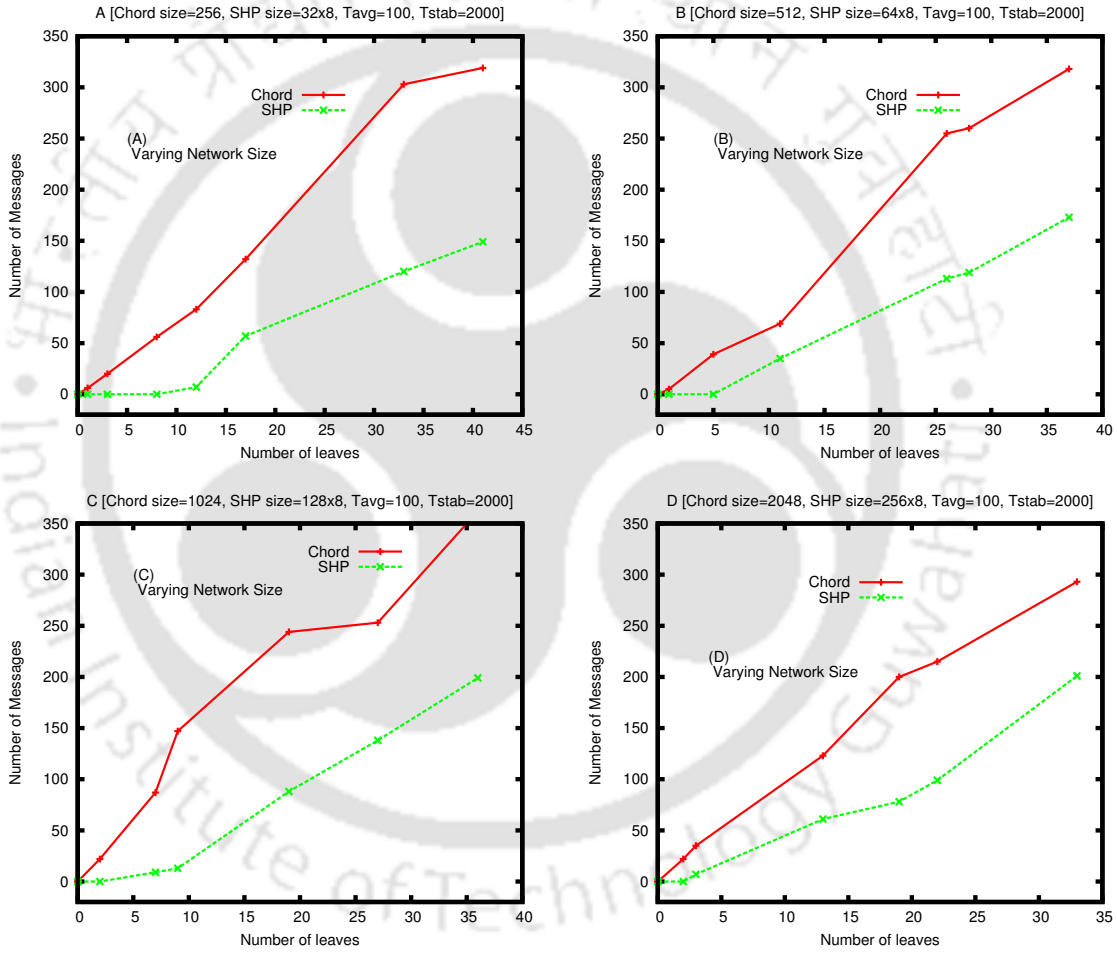

Figure 3.9: Graph showing number of messages required to leave nodes in Chord and SHP with different network size

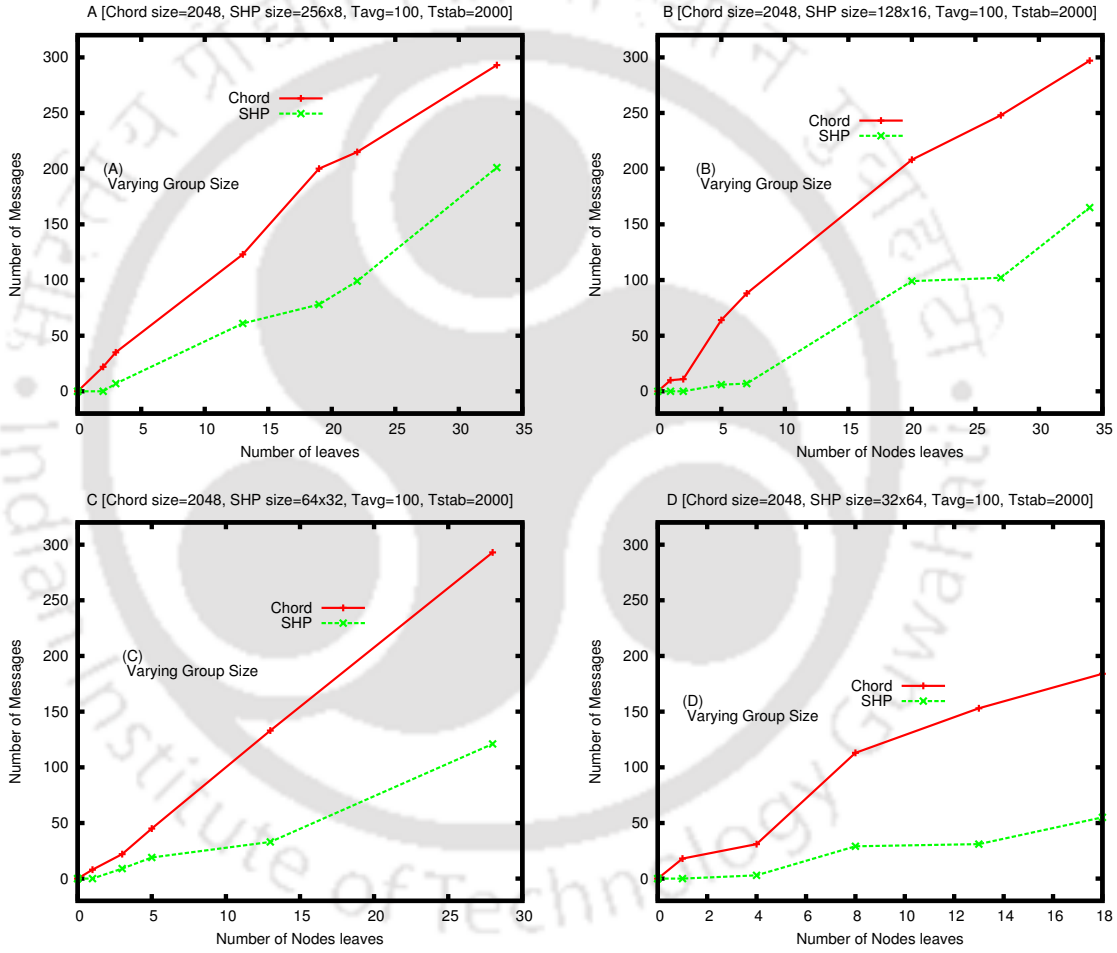

Figure 3.10: Graph showing number of messages required to leave nodes in Chord and SHP with different group size of same network size

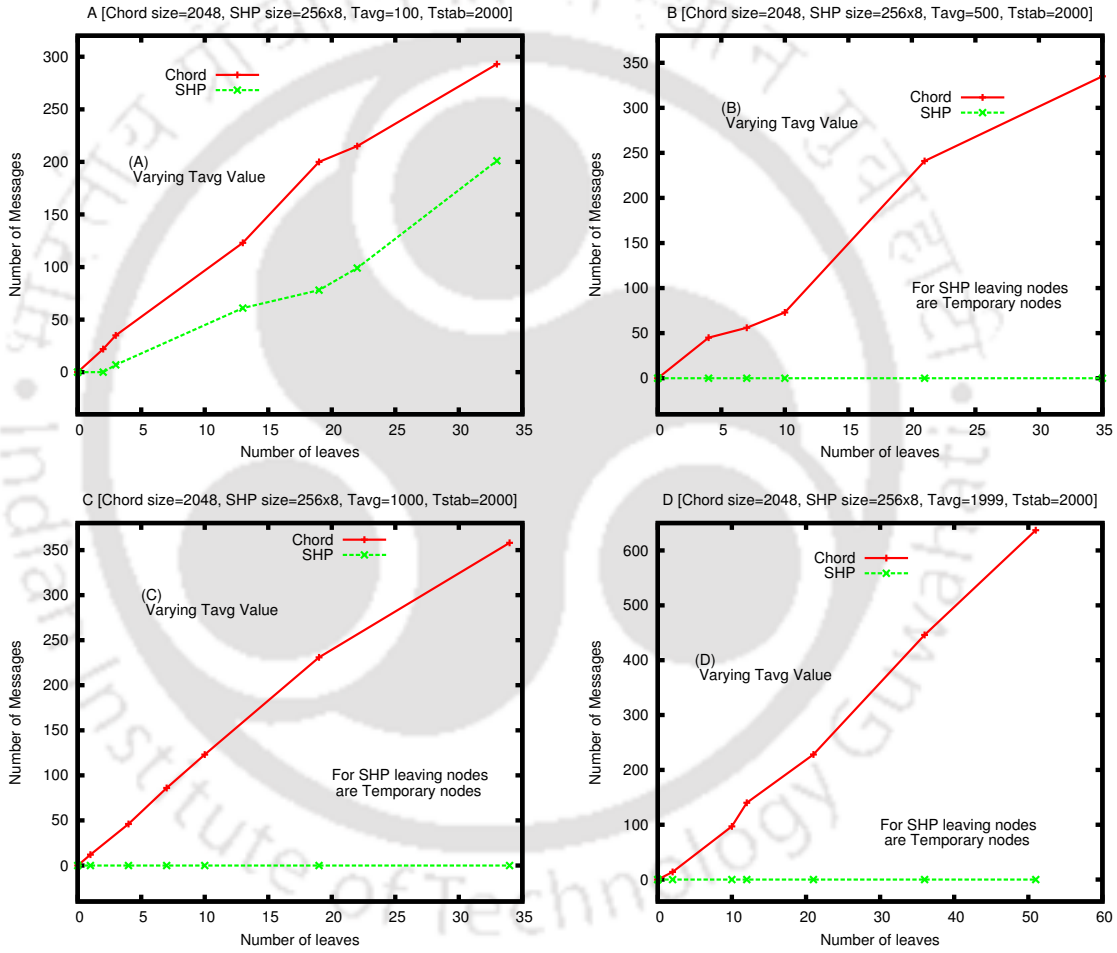

Figure 3.11: Graph showing number of messages required to leave nodes in Chord and SHP with different T_{avg} value of same network size

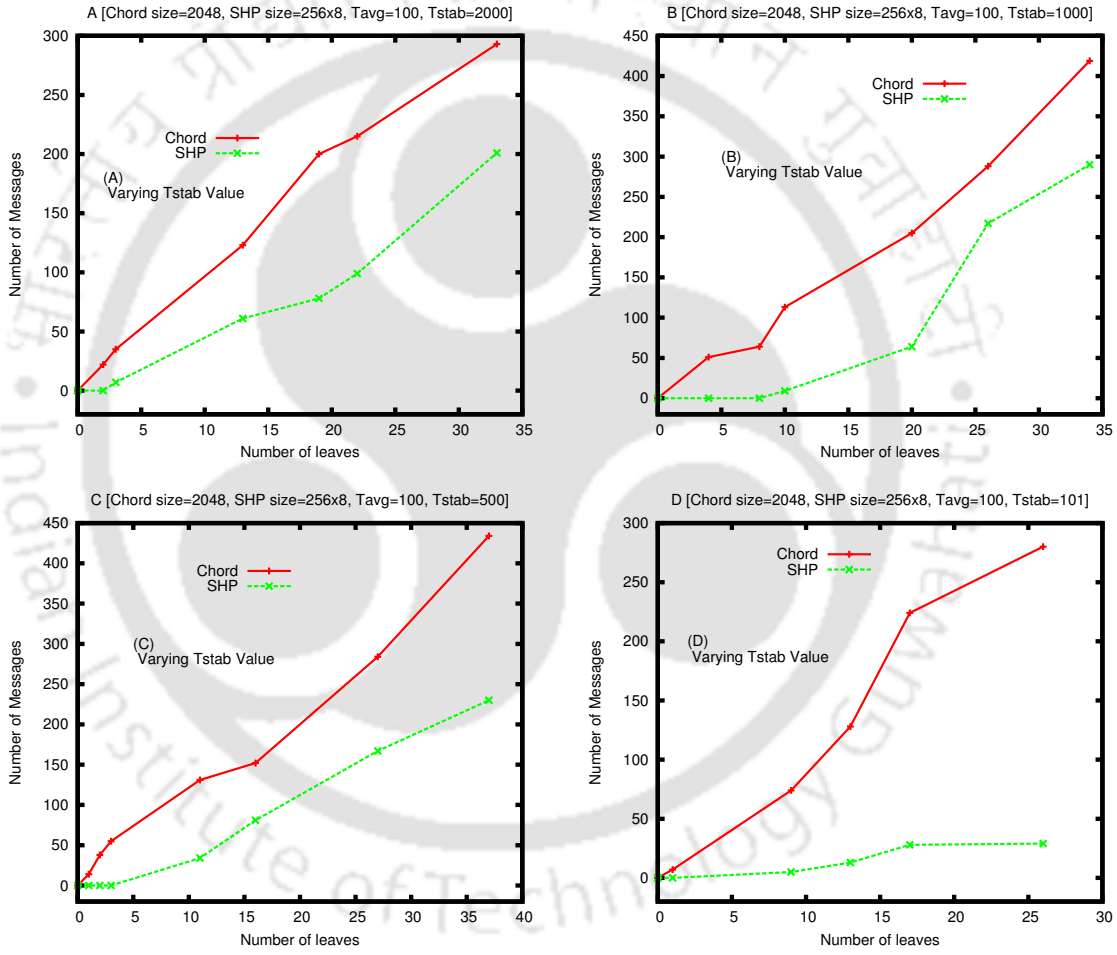

Figure 3.12: Graph showing number of messages required to leave nodes in Chord and SHP with different T_{stab} value of same network size

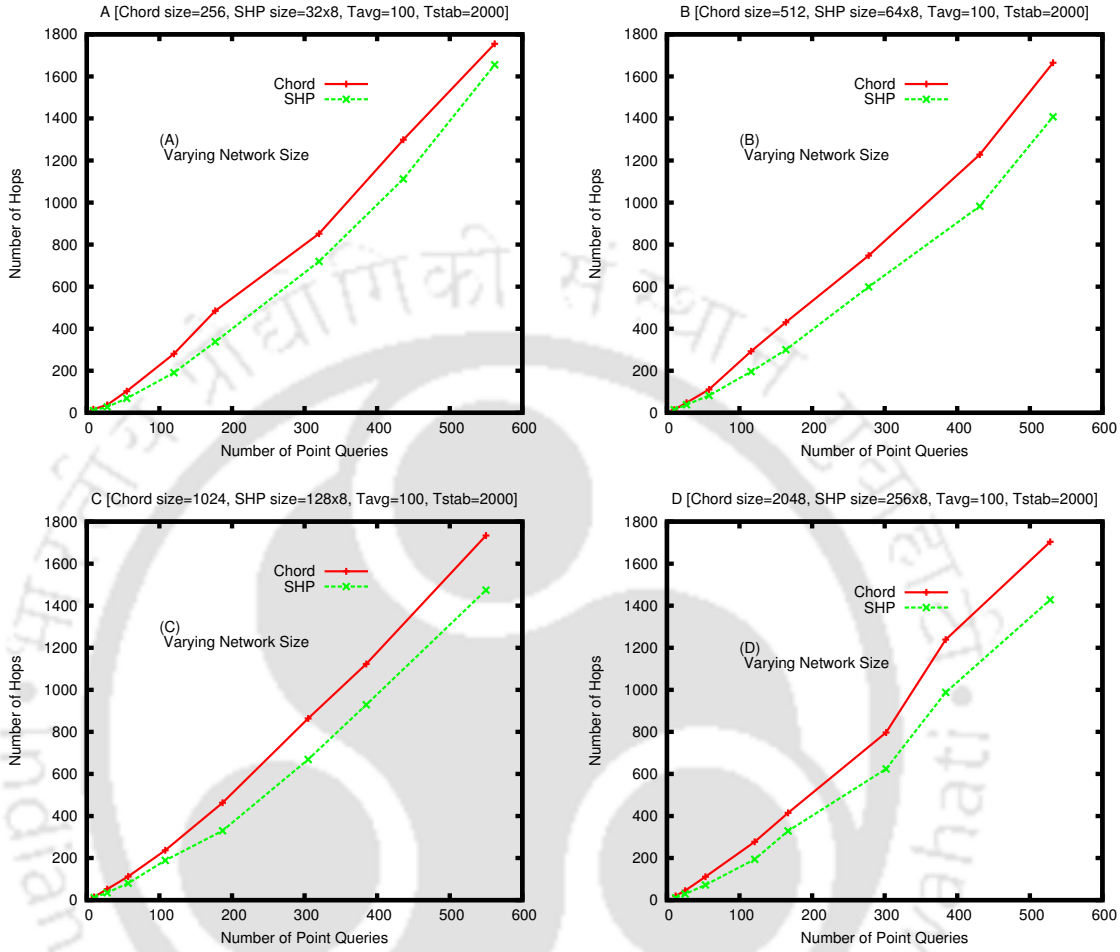

Figure 3.13: Graph showing number of hops required for point query in Chord and SHP with different network size

than Chord.

Figure 3.17, 3.18, 3.19, 3.20 and 3.21 illustrate the number of hops needed to execute range query in various configuration. In Figure 3.17, 3.18, 3.19 and 3.20, it is shown with various network sizes, changing the group size and different T_{avg} and T_{stab} values keeping range size fix at 10. Different range sizes such as 10, 20, 30, 40 are illustrated in Figure 3.21. In all cases performance improvement is observed significantly for range query in SHP.

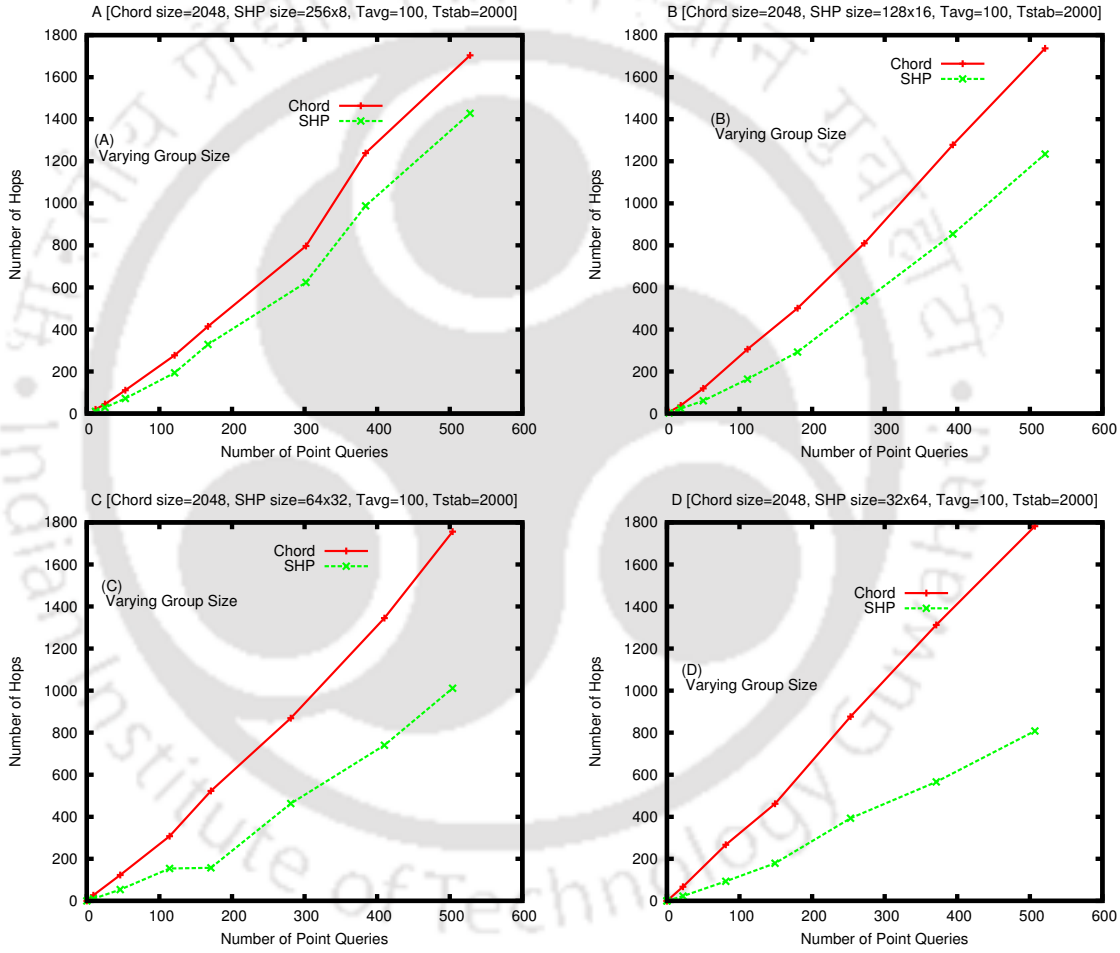

Figure 3.14: Graph showing number of hops required for point query in Chord and SHP with different group size of same network size

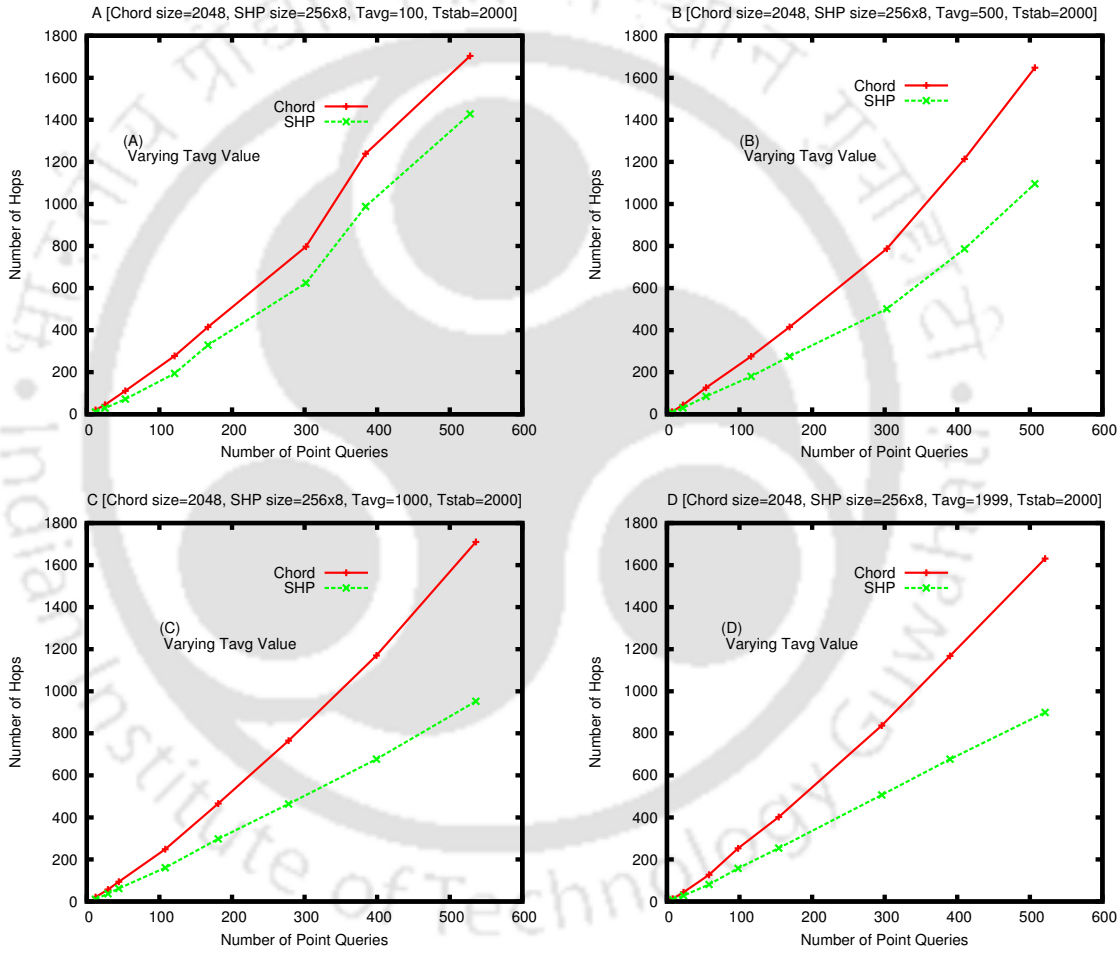

Figure 3.15: Graph showing number of hops required for point query in Chord and SHP with different T_{avg} value of same network size

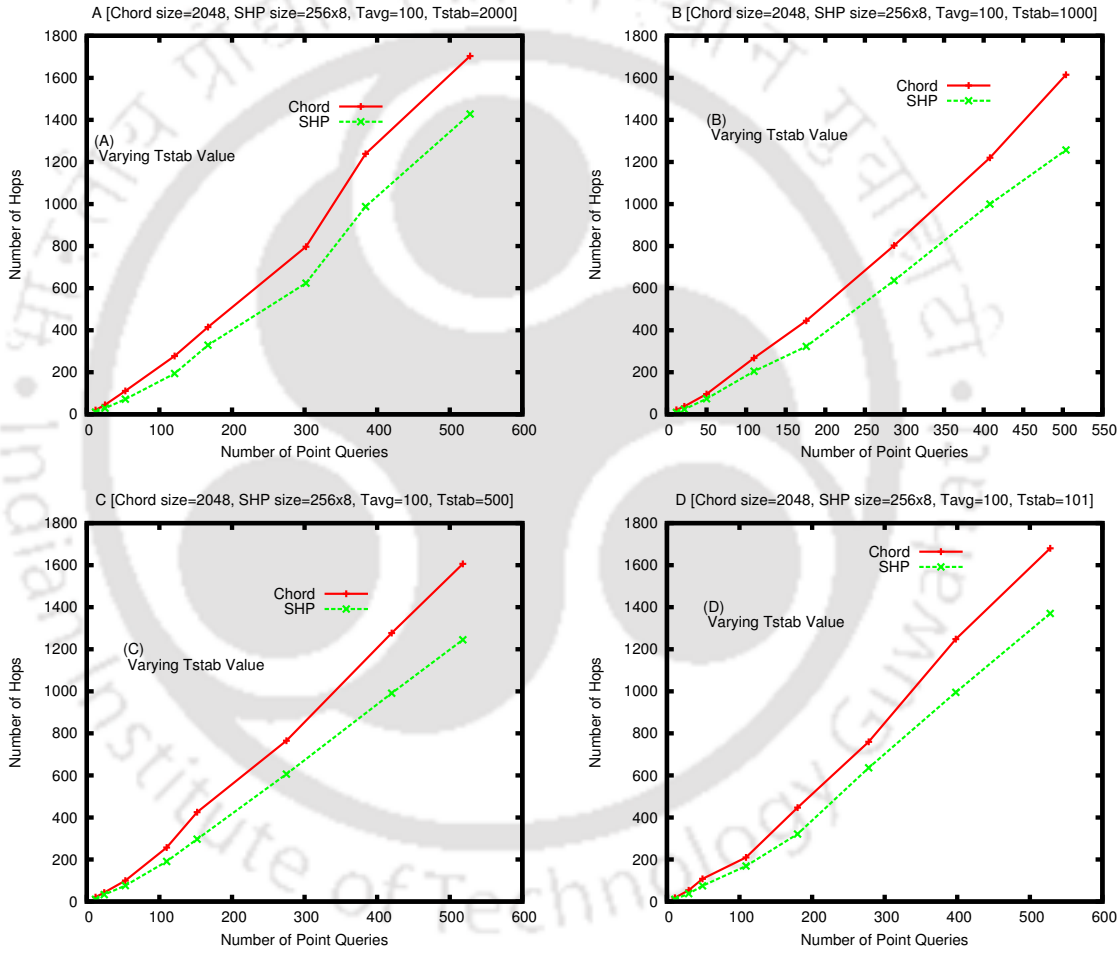

Figure 3.16: Graph showing number of hops required for point query in Chord and SHP with different T_{stab} value of same network size

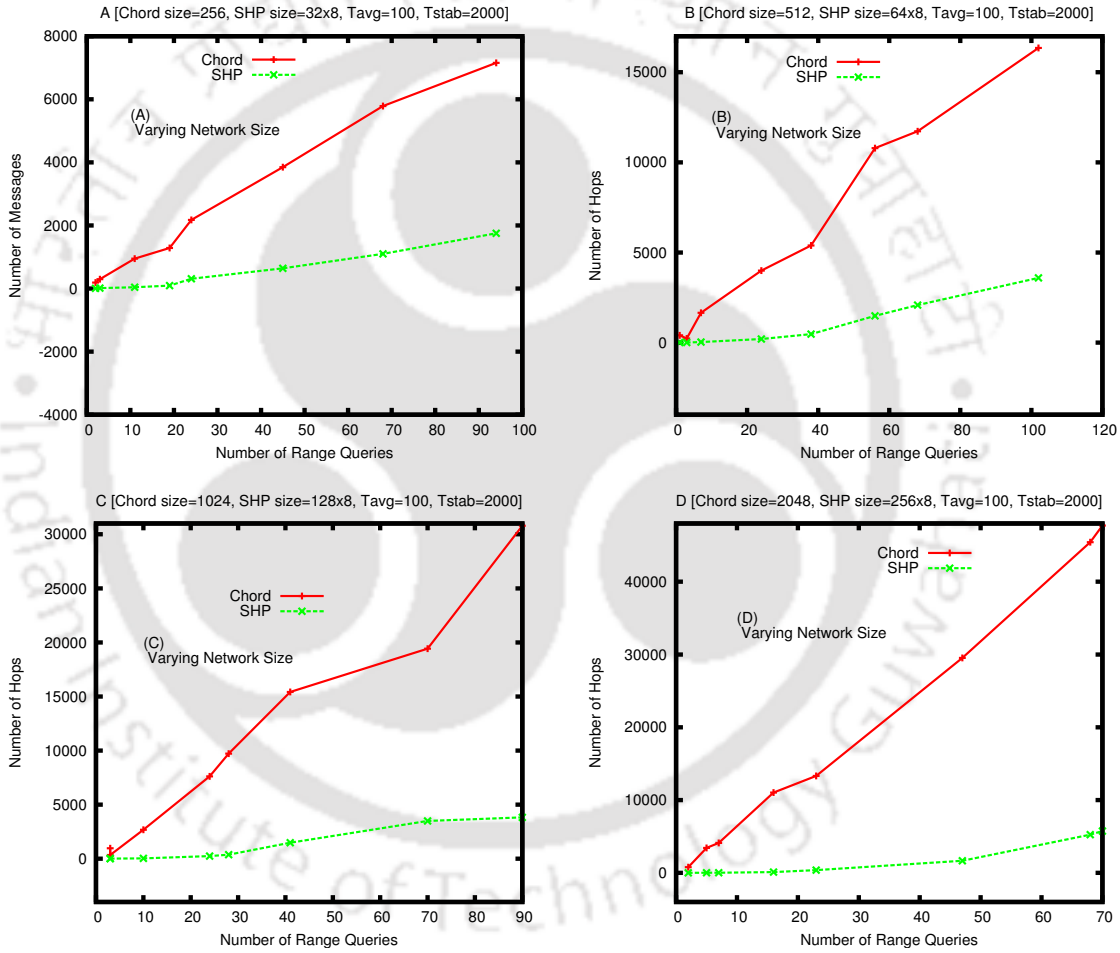

Figure 3.17: Graph showing number of hops required for range query in Chord and SHP with different network size

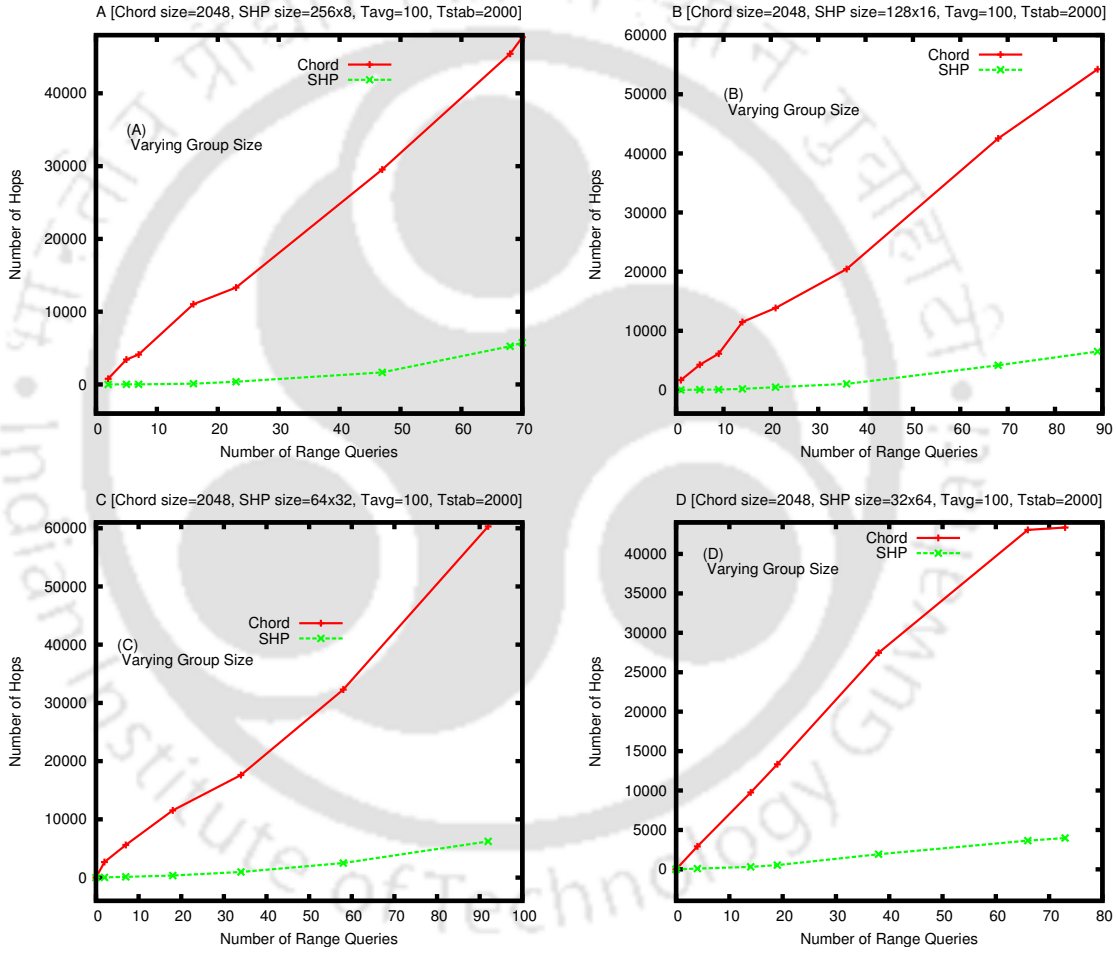

Figure 3.18: Graph showing number of hops required for range query in Chord and SHP with different group size of same network size

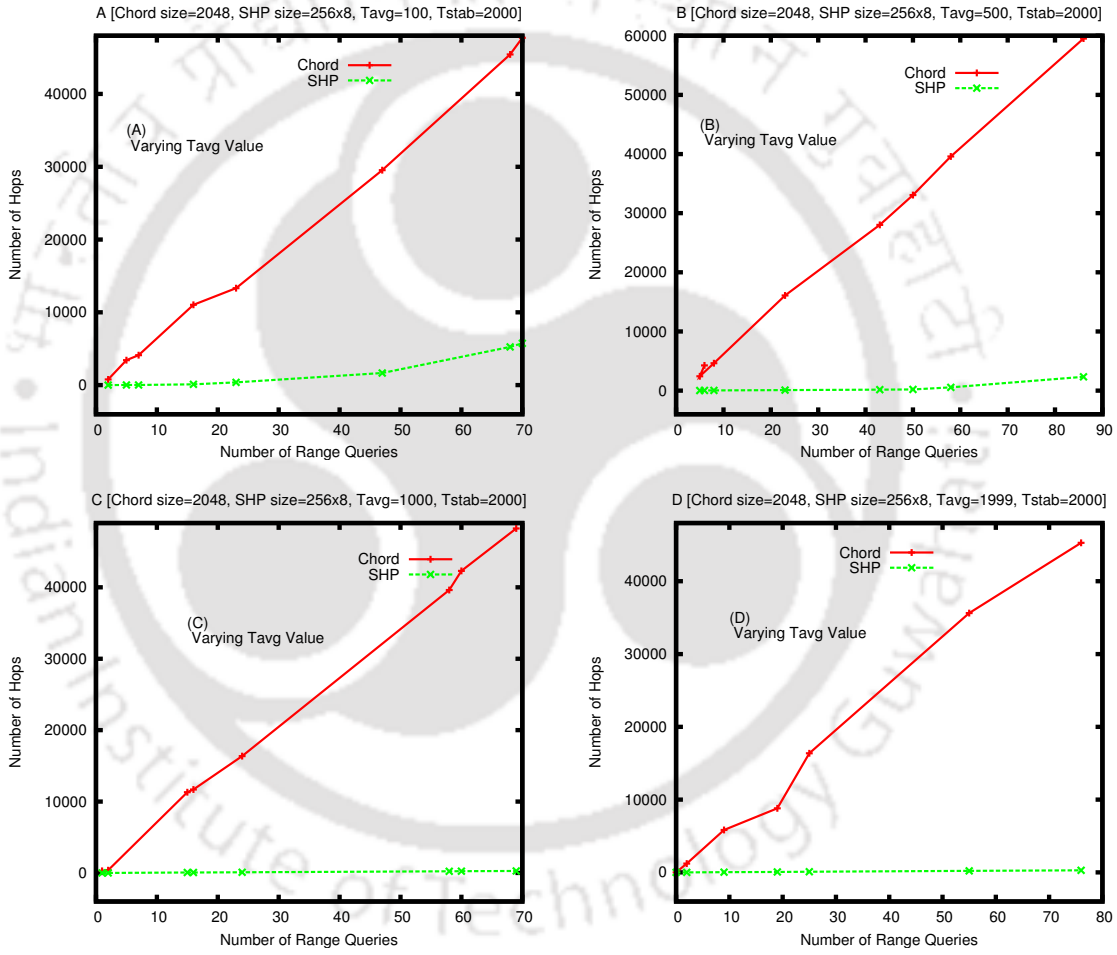

Figure 3.19: Graph showing number of hops required for range query in Chord and SHP with different T_{avg} value of same network size

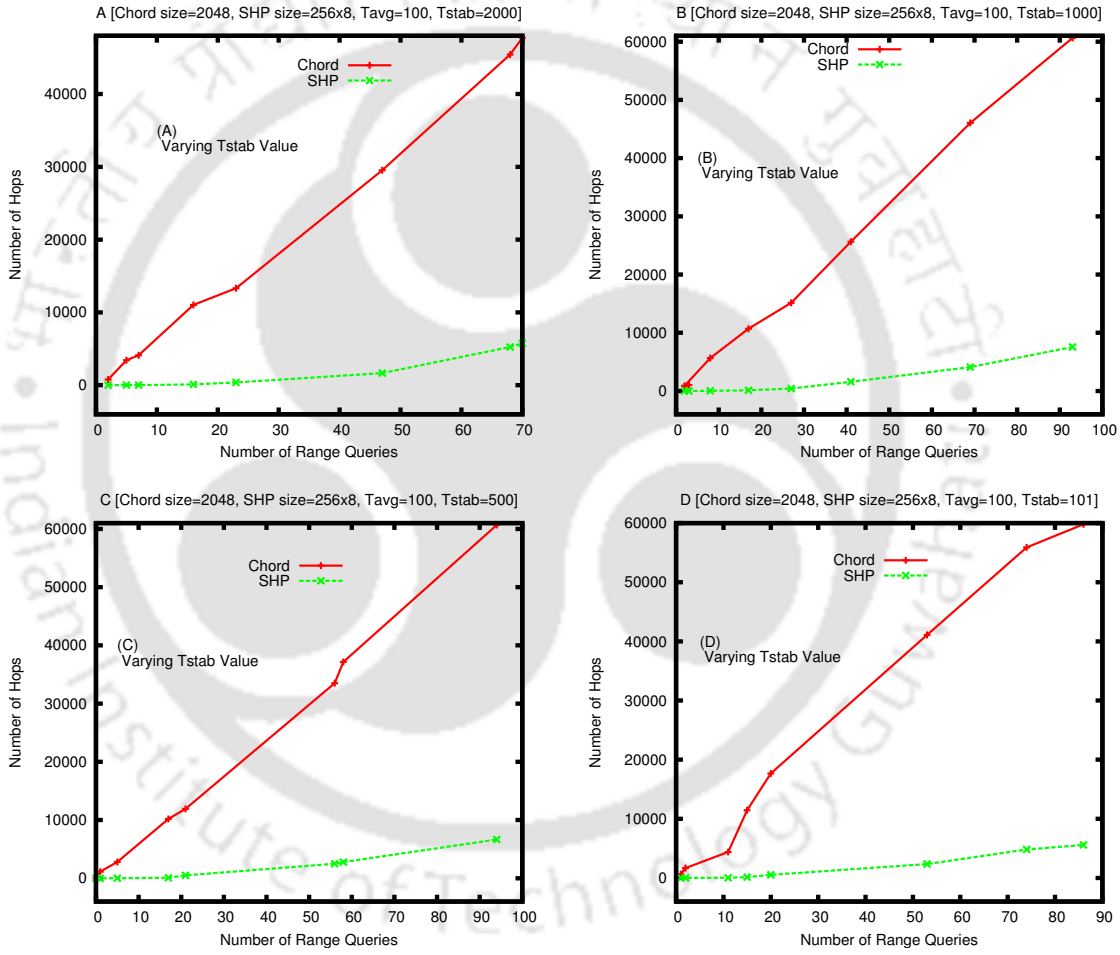

Figure 3.20: Graph showing number of hops required for range query in Chord and SHP with different T_{stab} value of same network size

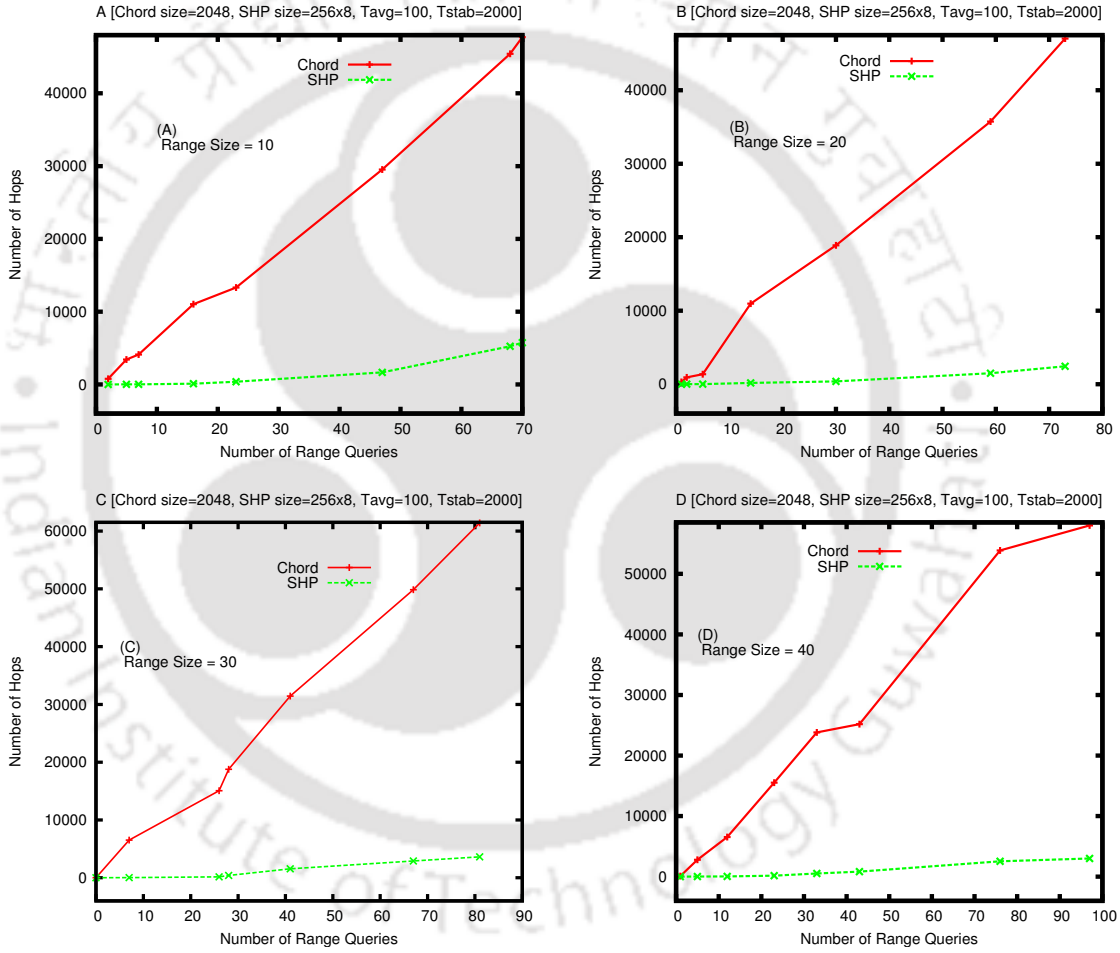

Figure 3.21: Graph showing number of hops required for range query in Chord and SHP with different range value of same network size

3.7.4 Load Balance

The load balancing mechanism is applied to distribute loads among nodes of the system. The average data of a node is the ratio of the total data space in the system to the total nodes (stable and fully-stable) in the system. The load factor of each node is the ratio of data stored in each node to the average data of a node. The grade value of nodes are shown in Table 3.7 which represents the situation of the system before load balance and after load balance. Figure 3.22, 3.23, 3.24 and 3.25 represent the grade values of nodes before load balancing and after load balancing. It is observed that loads are distributed in all nodes and the grade values of heavily loaded nodes are decreased and most of the nodes become normal loaded nodes.

3.7.5 Storage Requirement

It is seen from the Table 3.5 that the storage requirement in SHP is increasing as the number of fully-stable node increases. The same is represented in Table 3.5. Figure 3.26, 3.27, 3.28 and 3.29 illustrate the table sizes required in SHP and Chord for same number of nodes join. It is shown with varying sizes of network, varying sizes of groups, different values of T_{avg} and T_{stab}. In Figure 3.26, it is observed that storage requirement of SHP is significantly less for large difference between the number of nodes in a group and number of groups in the system. This is because in that case number of fully-stable nodes are less than temporary and stable nodes.

Figure 3.27 illustrates the effects of number nodes in a group and number of groups available in the system. It is found that as the number of groups in a system increases and the number of nodes in a group decreases, the population of fully-stable nodes in the system become higher. Therefore, the storage requirement for SHP increases as shown in Figure 3.27-[C] and 3.27-[D].

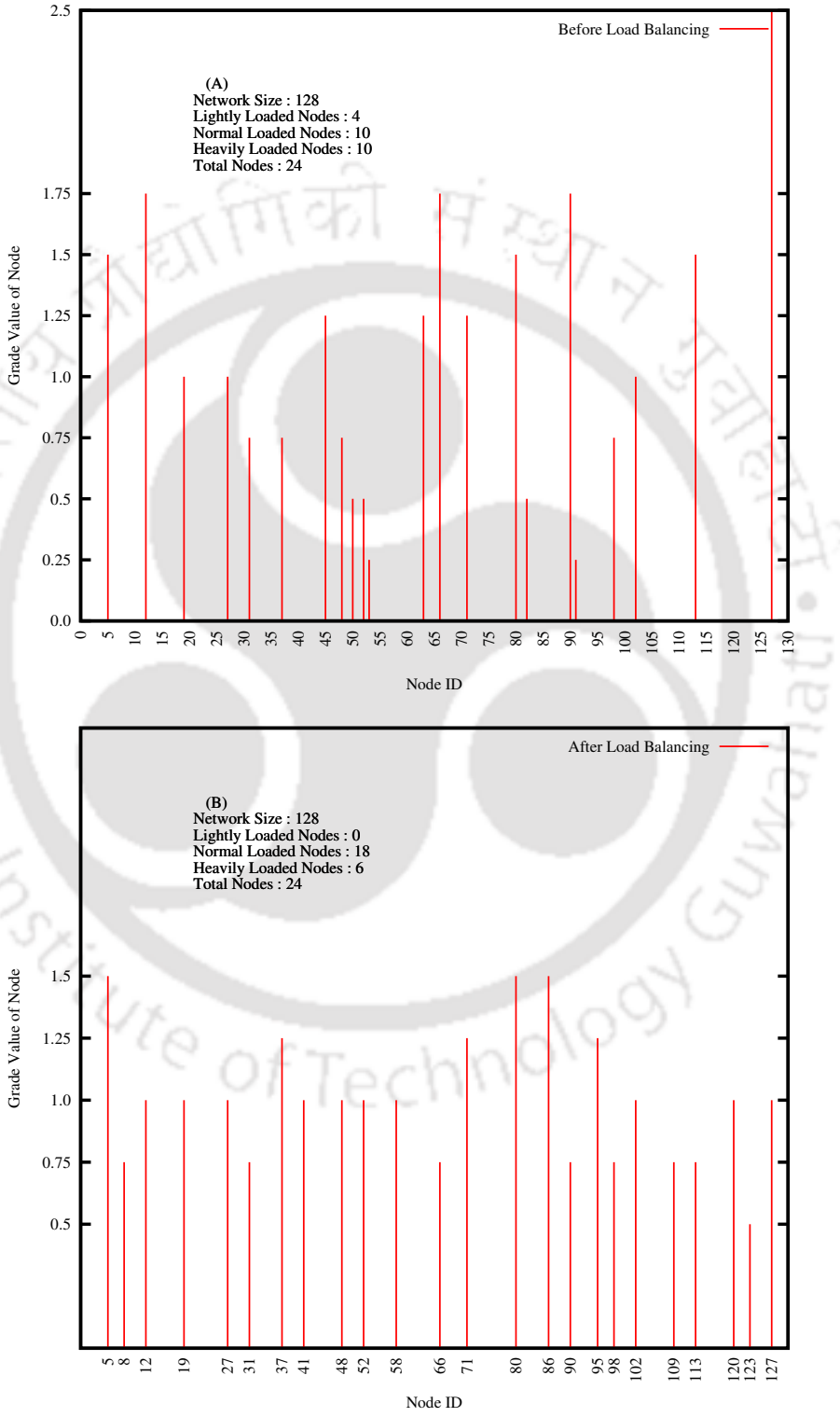

Figure 3.22: Graph showing load balance in SHP with 100 events

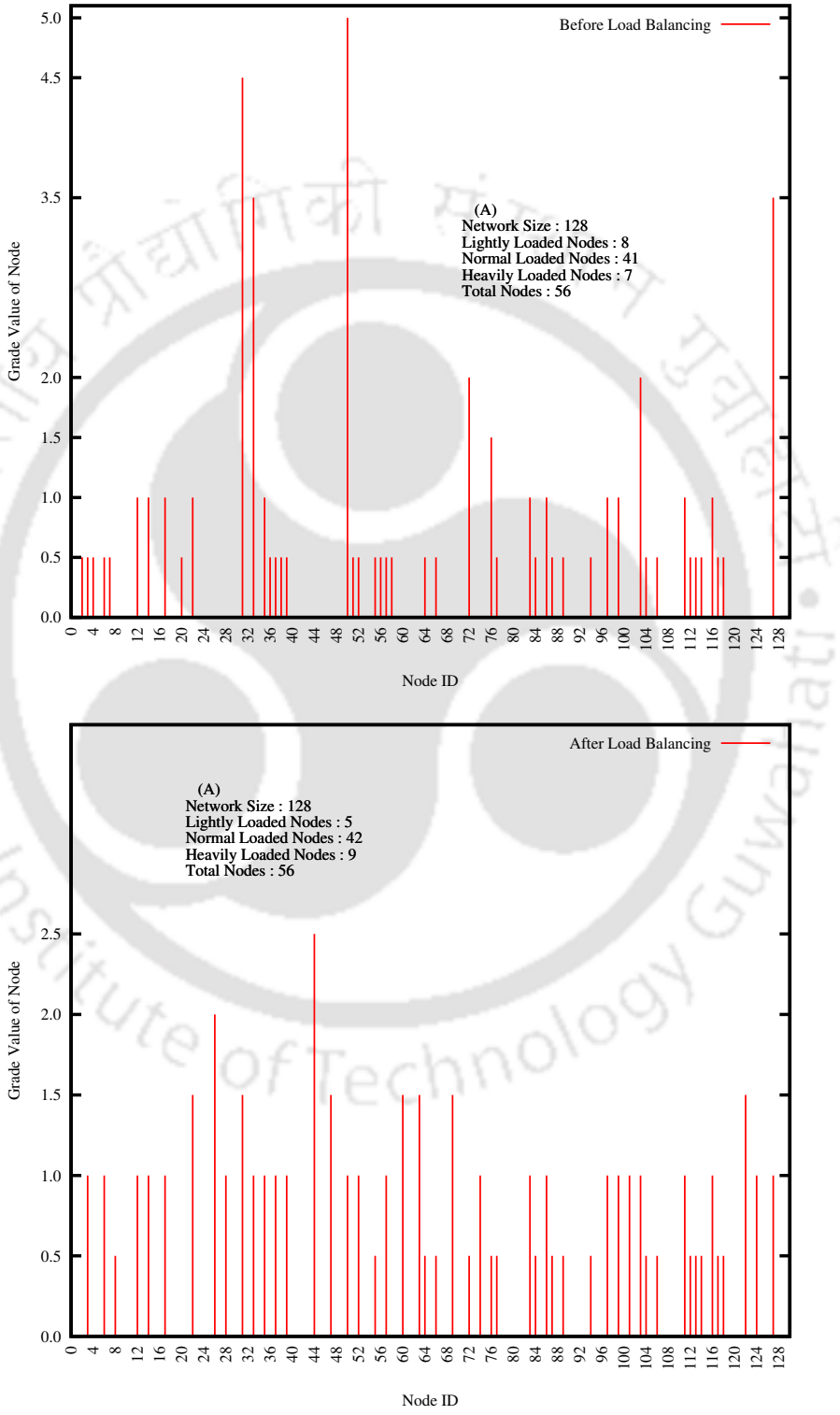

Figure 3.23: Graph showing load balance in SHP with 300 events

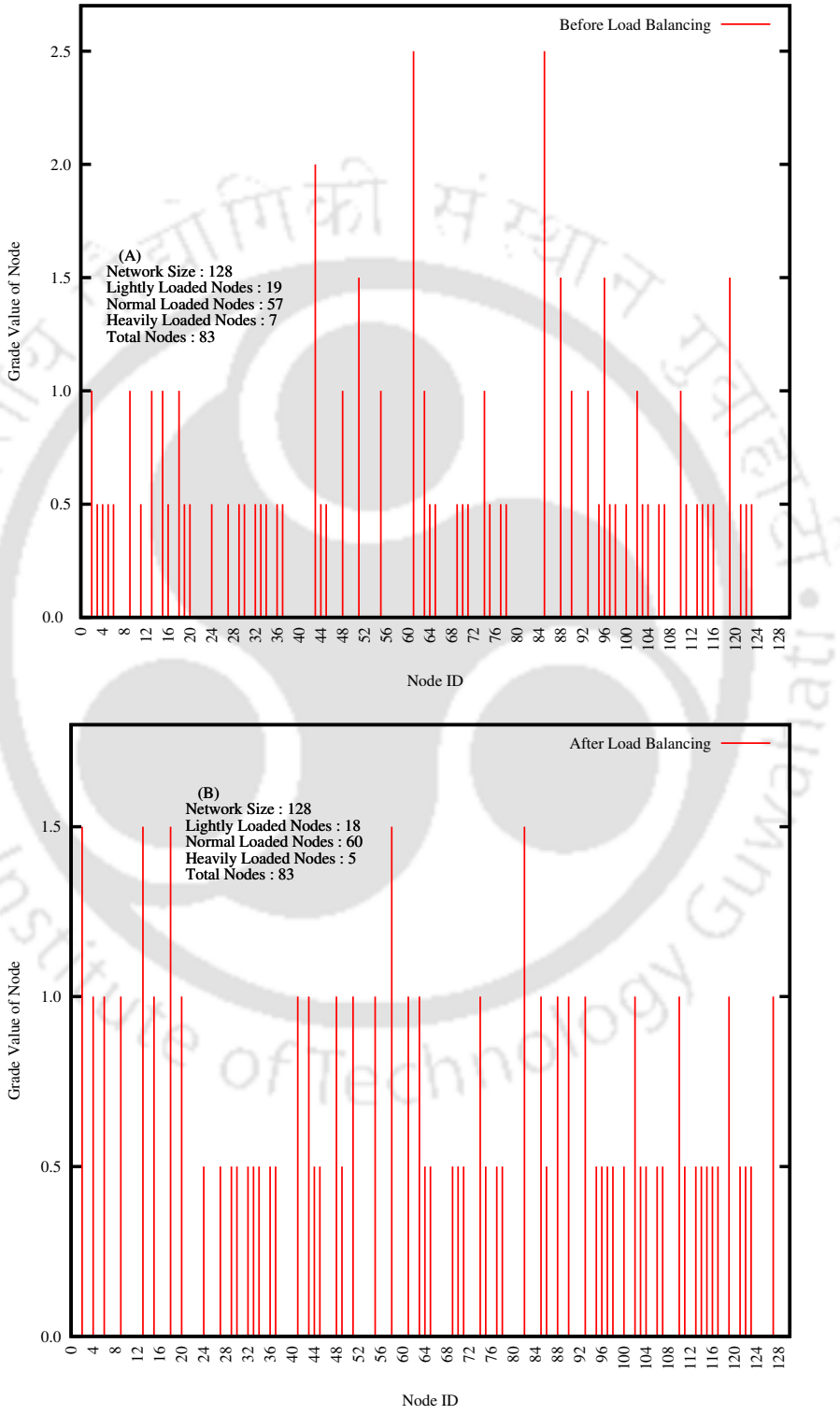

Figure 3.24: Graph showing load balance in SHP with 500 events

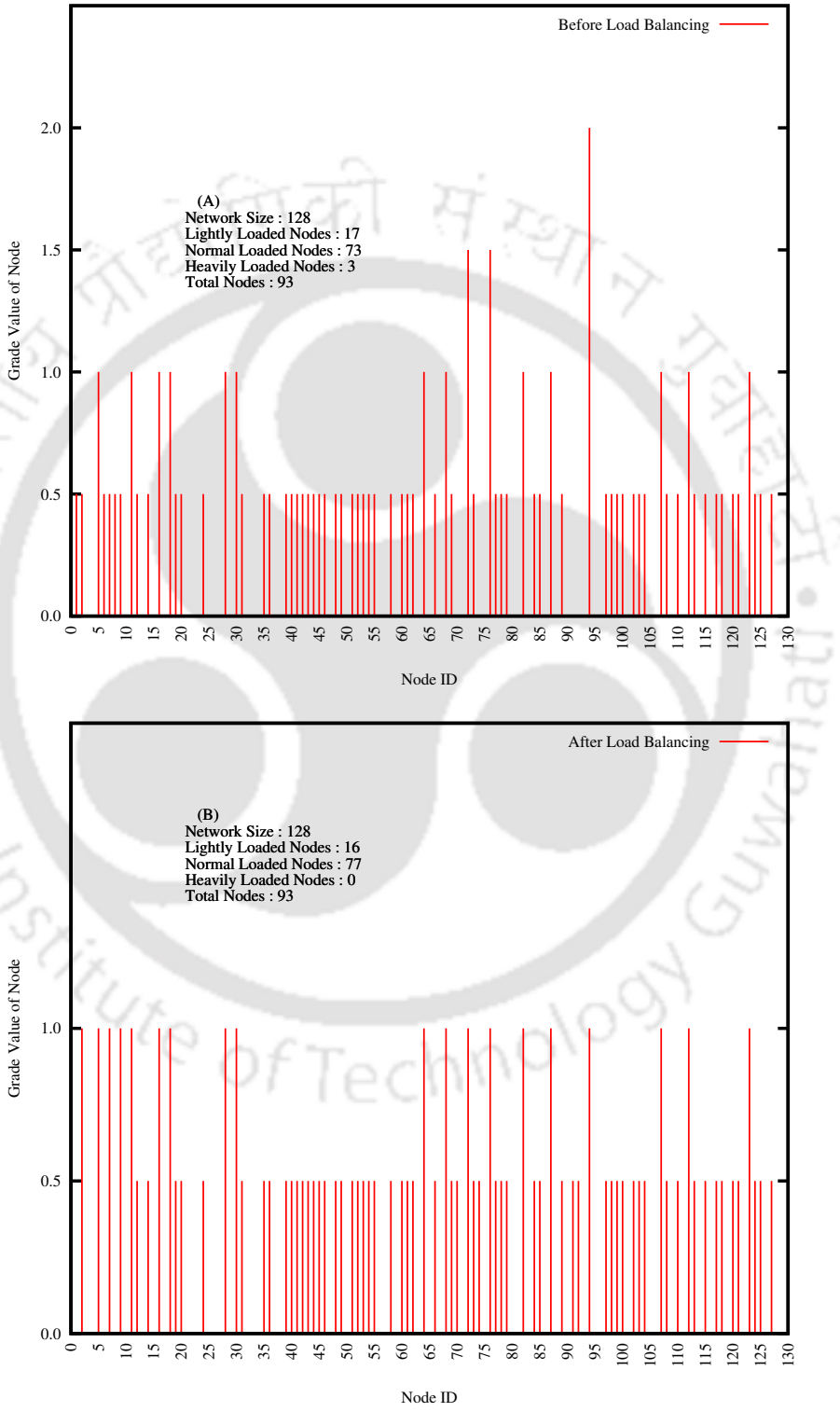

Figure 3.25: Graph showing load balance in SHP with 700 events

Before Load Balance			After Load Balance		
Node ID	Data Available	Grade Value	Node ID	Data Available	Grade Value
5	6	1.5	5	6	1.5
12	7	1.77	8	3	0.75
19	4	1.0	12	4	1.0
27	4	1.0	19	4	1.0
31	3	0.75	27	4	1.0
37	3	0.75	31	3	0.75
45	5	1.25	37	5	1.25
48	3	0.75	41	4	1.0
50	2	0.5	48	4	1.0
52	2	0.5	52	4	1.0
53	1	0.25	58	4	1.0
63	5	1.25	66	3	0.75
66	7	1.75	71	5	1.25
71	5	1.25	80	6	1.5
80	6	1.5	86	6	1.5
82	2	0.5	90	3	0.75
90	7	1.75	95	5	1.25
91	1	0.25	98	3	0.75
98	3	0.75	102	4	1.0
102	4	1.0	109	3	0.75
105	0	0.0	113	3	0.75
113	6	1.5	120	4	1.0
114	0	0.0	123	2	0.5
127	10	2.5	127	4	1.0

Table 3.7: Grade value of nodes for *SHP size*=128 and *Number of stable nodes*=24

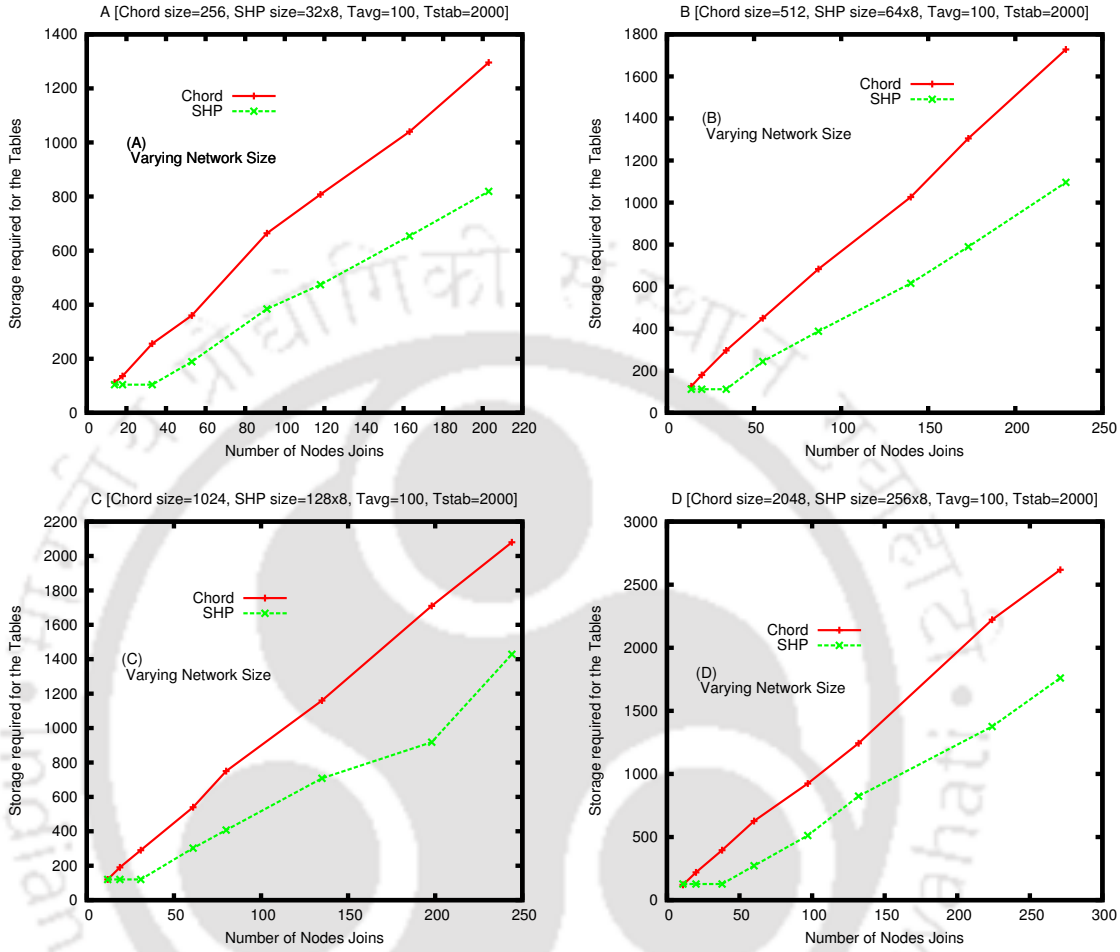

Figure 3.26: Graph showing size of the Tables used in Chord and SHP for different network size

The number of stable nodes decreases and number of temporary nodes increases in the system as T_{avg} increases. Hence the storage requirement reduces in the system as illustrated in Figure 3.28.

The number of fully-stable nodes increases in the system as the value of T_{stab} approaches the value of T_{avg}. Hence the storage requirement of the system increases as shown in Figure 3.29-[D].

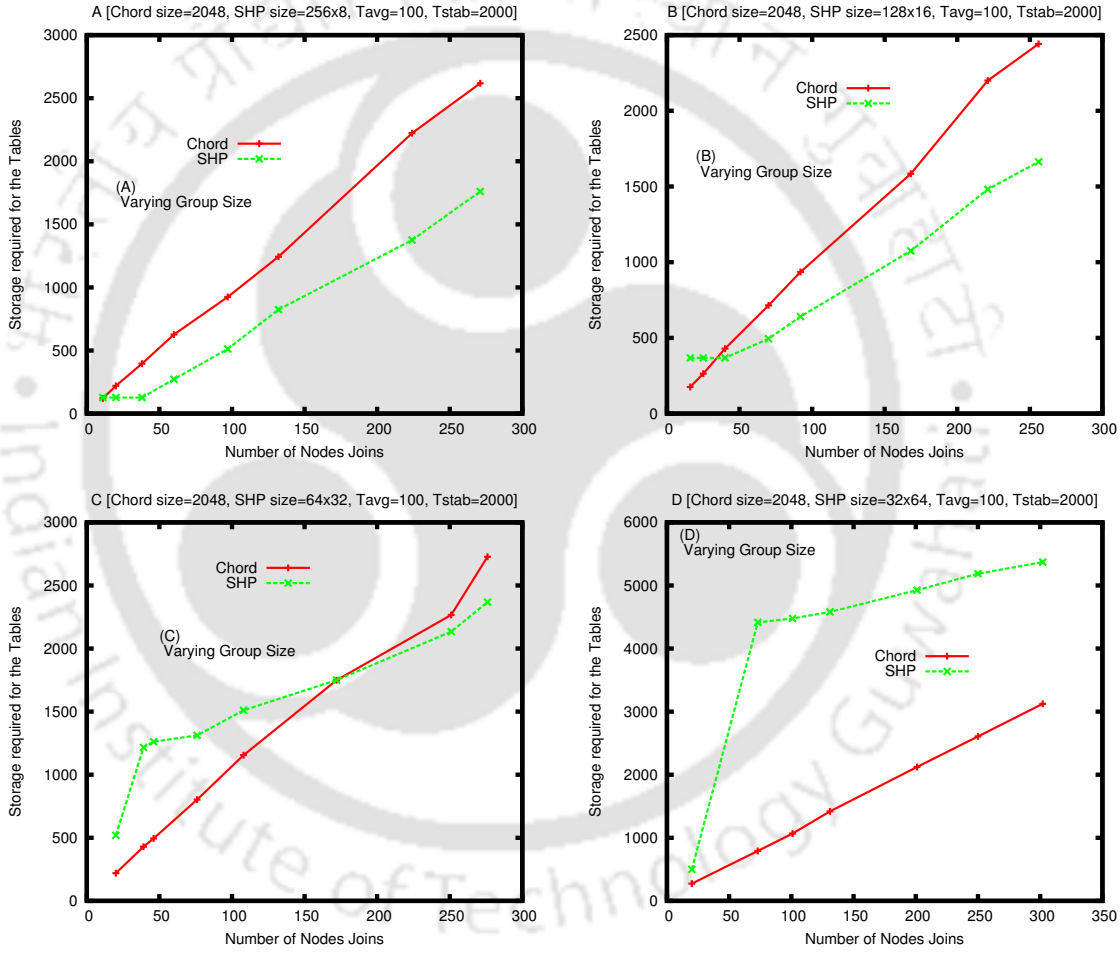

Figure 3.27: Graph showing size of the Tables used in Chord and SHP for different group size of same network size

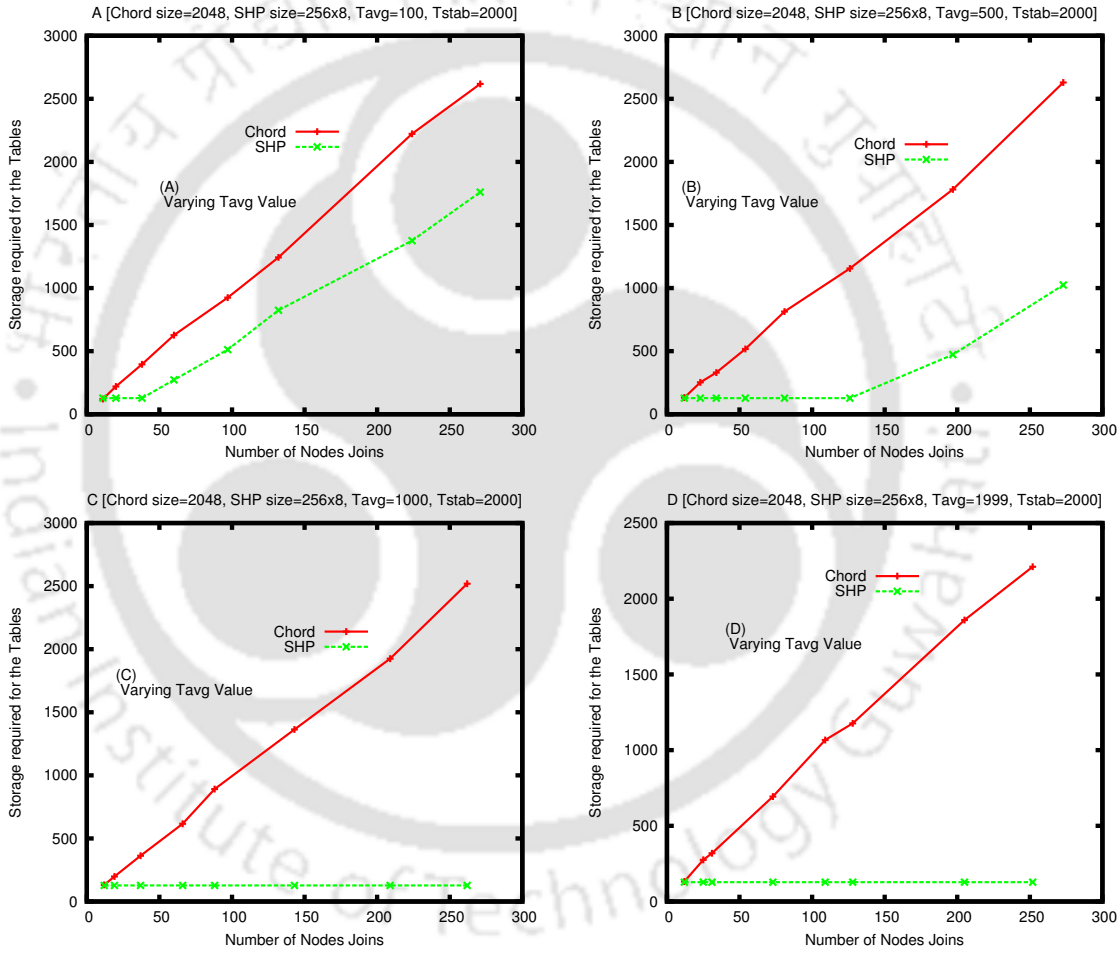

Figure 3.28: Graph showing size of the Tables used in Chord and SHP for different T_{avg} value of same network size

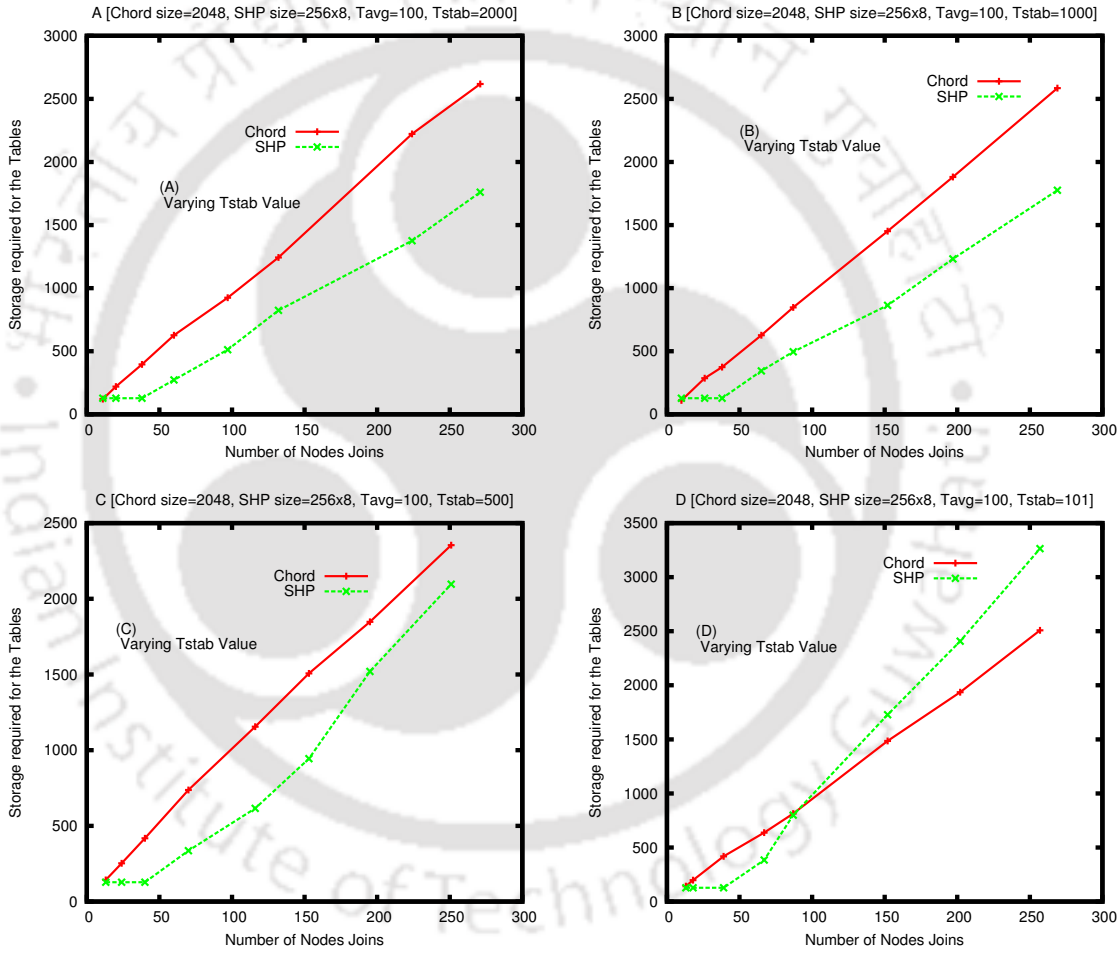

Figure 3.29: Graph showing size of the Tables used in Chord and SHP for different T_{stab} value of same network size

3.8 Related Works

In [GEBF+03], Garces-Erce et al. organize peers into groups in their general tier hierarchy of p2p networks. Top level overlay is a Chord ring but each group has its autonomous intra-group overlay network, which depends upon the number of nodes in the group. This system does not discuss the issue of transient node population. Ratnasamy et al. [RHKS02] present a binning scheme in which nodes that fall within a bin are relatively close to each other, hence reduces the latency. SHP uses the Round Trip Time (RTT) messages to find out the group to whom node belongs according to the network proximity. Lin et al. [LYHL06] exploit the p2p hierarchical overlay structure to present a fault-tolerant approach for p2p file sharing system. They have introduced superpeer design where superpeer acts as a centralized server to manage a group of peer nodes. Li et al. [LTZ08] analyze the promising supernode based p2p network structure. The scale of a p2p network plays an important role in determining network performance and they have integrated two important operational issues of a p2p network: sizing and grouping decisions. Wang et al. [WXZ05] organize the nodes in a 2-tier hierarchical structure. They propose an efficient query answering mechanism, but do not consider the churn rate problem, which may increase the traffic load and degrade overall performance. Zols et al. [ZSKT05] propose HCP (Hybrid Chord Protocol) which supports the grouping of shared objects according to interest. Since they do not consider hierarchical structure, lookup and query routing still is a problem. Gao et al. [GS04] describe an adaptive system where a logical tree is maintained to implement range query. Temporary nodes are an integral part of this protocol which increases the cost of maintaining the tree. Marzolla et al. [MMO06] describe a data location strategy for dynamic content on p2p networks. Data location exploits a distributed index based on bit vectors. They propose a protocol that allows peers to locate data matching range queries i.e. queries that search for all data items whose values fall into a given interval. Ntarmos et al. [NT04] use a second Chord ring over and above the conventional Chord system. The

second level called the range guard takes advantage of the more powerful nodes in the system. In this approach routing table maintained for every node is unnecessary because temporary nodes do not share anything and do not remain in the system for a long time. The SHP maintains routing information only for stable and fully-stable nodes. In [SKG07], Srivastava et al. describe query processing using Chord. In SHP, we have implemented range query and point query with better performance than Chord. Gupta et al. [GAA03] use locality sensitive hashing to hash similar data partitions to nearby identifiers. This approach is vulnerable to overloaded nodes due to skewed data partitions. Li et al. [LX06] use a distributed load balancing algorithm based on virtual servers. Each node aggregates load information of the neighbors and moves the virtual servers from heavily loaded nodes to lightly loaded nodes. In Chord [SMLN+03], a solution for load-balancing is proposed using *virtual peers*. Though it is a simple method for load-balancing, it is not efficient due to hefty messaging cost. Rao et al. [RLS+03] propose three algorithms to rearrange load based on different capacities of nodes: one-to-one, many-to-many and one-to-many. Their basic idea is to move load from heavy nodes to light nodes so that each node's load does not exceed its capacity.

3.9 Summary

A new overlay called structured hierarchical protocol is introduced in this chapter. This system is shown to be capable of handling high churn rate. Another important feature of this system is that unlike existing structured systems, it can efficiently execute both point and range queries. A very simple load-balancing mechanism for even distribution of loads among the nodes is proposed. It has been observed that the system is highly efficient with respect to storage requirement. The Chord is considered as basis for comparison since Chord is well-defined and its functionalities has already been proved.

Chapter 4

Efficient Handling of High Churn Rate

A p2p system has the characteristics of local control of data, dynamic addition and removal of nodes, local knowledge of available data and schema, self-organization and self-optimization. The nodes may be unreliable as most of nodes join p2p system only for their needs, e.g. for searching and downloading files, which is on average only one hour [SGG02]. The churn rate is defined as the ratio of the number of participants who discontinue their use of a service to the average number of total participants for any given period of time. Churn rate provides insight into the growth or decline of the users base as well as the average length of participation in the service. The effect of churn in Chord [SMLN$^+$03] is clearly visible with reduced performance. Chord needs only $O(\log n)$ messages for lookup, where n is the total number of nodes in the system. But $O(\log n)^2$ messages are needed for a node to maintain its routing table during joining and leaving. It is known that higher dynamic of the network implies the shorter interval to refresh routing tables so higher the maintenance cost.

A modification to the Chord protocol is presented to handle churn rate which in turn improves the efficiency of the system. Classification of nodes is done

in section 4.1 and description of the protocol is presented in section 4.2. Section 4.3 gives the experimental results and related works have been discussed in section 4.4. The summary is presented in section 4.5.

4.1 Node Classification

Nodes in a network are differentiated based on their time for which they are in the network. The nodes available in the system is classified as (i) temporary node, (ii) quasi-stable node and (iii) fully-stable node according to their time span in the system as shown in Figure 4.1. The nomenclature of nodes are different from chapter 3 to distinguish the function of stable node and quasi-stable node.

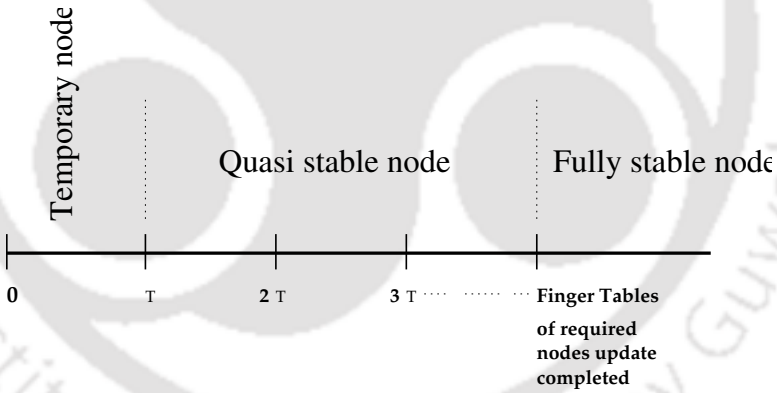

Figure 4.1: Illustration of nodes according to time scale

1. **Temporary nodes:** These are recently joined nodes which are connected to the network via a fully-stable node. They are not exposed to the Chord network up to time unit τ.

2. **Quasi-stable node:** A node is called a quasi-stable node if it is in the network more than τ time units. A quasi-stable node is expected to be in the network for more time. The finger table entries of all other nodes which should have the information of this node are not yet updated.

3. **Fully-stable nodes:** A quasi-stable node becomes a fully-stable node as soon as all entries of the finger table are updated. A fully-stable node is expected to be in the network for long time.

4.2 The Protocol

A concept called *Temporal Centralization* protocol (TCP) is proposed to reduce the number of joining for transient nodes. Figure 4.2 shows the overlay network with different type of nodes. During bootstrapping period of the system, $\lceil \log n \rceil$ nodes join the overlay network in conventional manner to become fully-stable node where n is the size of data space. After bootstrapping, the node joins in the second level of hierarchy as temporary node. In this model, each node maintains a separate routing table apart from the regular finger table called *Temporary Routing Table* (TRT) as shown in Table 4.1.

NODEID	TIME STAMP	QUERY SPACE
19	τ	Request for file with Key *id* 46

Table 4.1: Temporary Routing Table

This TRT maintains the entries corresponding to the temporary nodes. A temporary node joins the network through a fully-stable node. Each fully stable node keeps information about $\log n$ nodes. When a temporary node joins the network, it will remain as temporary up to τ time unit. The value of τ is initialized as $\lceil \log n \rceil$ which is an empirically set parameter based on observation. The fully-stable node keeps information about the new node in the TRT. Query space in the TRT is used by the new node to make a query to the system and to store the response given by the system. The TRT is analogous to that of NAT (network address translation) or a proxy table. When a temporary node makes a

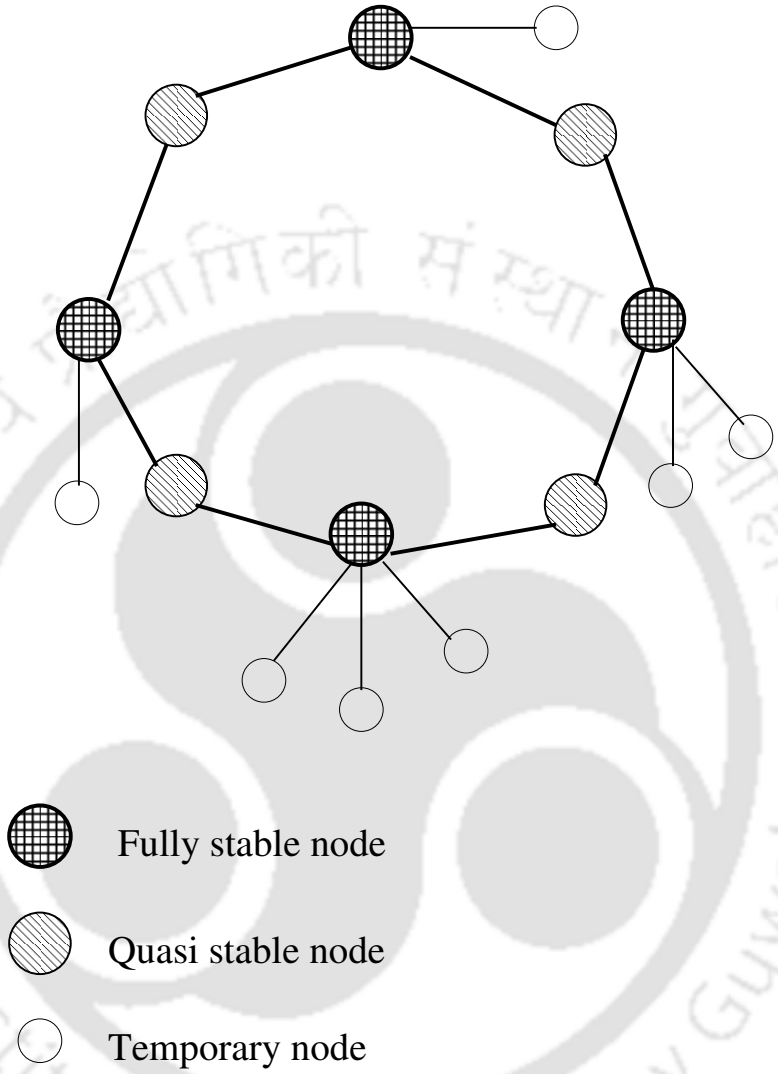

Figure 4.2: Illustration of Temporal approach

query it is forwarded by the fully-stable node to the system. After processing the query the message is stored in the TRT, which in turn forwarded to the temporary. If the temporary node leaves the network or fails then the TRT has to be updated and need not necessary to update the finger tables as it is not updated by quasi-stable and fully-stable nodes.

After τ time the temporary node initiates joining operation like in Chord.

Now the node is called a quasi-stable node. A finger table for the quasi-stable node is created. After another τ time it updates one entry of the finger table of each preceding node and so on until it completes updation of all the finger table entries to become a fully-stable node. If the quasi-stable node leaves the network at any point of time after updating some of the the finger table then updated nodes are rolled back to the state before the node has joined.

In this approach, upgradation of a temporary node to quasi-stable node requires $O(\log n)$ messages in first step. Then the system waits for τ time and updates other finger table entries gradually. Therefore, to initiate each join operation number of message required is $O(\log n)$. It is mentioned in Chord that finding and updating all nodes during joining takes $O(\log n)^2$. In Chord protocol, addition / leaving of a transient node causes updation of all the existing nodes in the network. In this approach, only a few of the nodes are updated and they can be rolled back to previous state on leaving or failure of the newly joined node. The Query processing remains same as that of the regular Chord protocol.

4.2.1 Node Joining

In a dynamic network, nodes can join or leave at any time. A node p joins the network by contacting any existing fully-stable node p', and asks p' to find successor of p in the ring. The algorithm for joining a new node through one existing fully-stable node is stated as follows:

- Step1: Find successor of the new node.
- Step2: Find predecessor of the successor node found in step1.
- Step3: Put new node as successor of the predecessor node found in step2.
- Step4: Initialize finger table of new node.

- Step5: Update finger table of other nodes preceding new node after a delay period of τ.

4.2.2 Performance Analysis

The performance of the system is highly dependent on the behaviour of temporary nodes. A temporary node joins and leaves from the system too frequently. As the churn rate of the system depends on τ, the performance analysis is done on the basis of τ time period.

Before τ time unit

A node available before τ is said to be temporary node. If a node joins or leaves during this period then it has no effect on the system. As described in section 3.3.1, this approach would save 66.66% of messages in comparison to conventional Chord protocol (CCP).

After τ Time Unit

In Chord protocol, whenever a node joins the network, the number of messages required is always $O(\log n)^2$. In Temporal centralization, upto τ period time number of update is zero and the updation of nodes started after τ. In the best case, only $\log n$ number of messages needed for lookup and update. So number of messages passed is $\log n$. In the average case the nodes are updated in steps of τ time units, where at each τ time unit step $(\log n)$ nodes are updated. So number of messages required is $O(\log n)$ in each step. In the worst case, the number of steps required to update all finger tables is $\log n$, hence number of messages required is $O(\log n)^2$. Therefore, after $\log n$ steps a node becomes fully-stable node and works as a regular node in Chord.

4.3 Experimental Results

The simulator developed for SHP as discussed in section 3.7.1 has been extended for TCP as well and named as *temporalsim*. The design and development of *temporalsim* is described in Appendix-*B*.

4.3.1 Simulation Environment

The relative performance of TCP and CCP is compared using same data space and equal number of nodes at any given instant. The simulation is done using different network size and varying average time (τ) value. The input data set is generated using different type of files as found in [ZSR06] for key values. The Table 4.2 shows the file appeared in a p2p system. The total key space is divided into 11 groups according to the percentage of file shared. For example, suppose the total data space consists of 100 Key *id*s, then *.mp*3 occupies 61.5%, *.jpg* occupies 7.54%, and so on.

To simulate the operations of TCP and CCP users need to supply number of bits of a node *id* (say m) and total number of events. Therefore, the network can have maximum of 2^m number of nodes having node *id*s from 0 to (2^m-1). The number of bootstrapping nodes for TCP is m. The bootstrapping nodes are added in TCP to form the initial overlay. During bootstrapping period nodes are joined in the overlay like nodes join in CCP. Therefore, messages required to join nodes are equal in TCP and CCP as shown in graphs of Figure 4.3-[A], 4.4-[A], 4.5-[A] and 4.6-[A]. After bootstrapping period is over, the TCP executes different type of operations as generated by the event file. The events considered in this simulation are (i) joining a new node, (ii) leaving of a node and (iii) searching i.e. execution of point query.

Sl no.	File types	Percentage of occurrence
1	mp3	61.5
2	jpg	7.54
3	gif	3.14
4	wma	2.70
5	htm	2.69
6	exe	2.65
7	wmv	2.62
8	mpg	1.91
9	wav	1.86
10	txt	1.62
11	Others	11.80

Table 4.2: Percentage of different file types shared in p2p system

4.3.2 Observations

Graphs in Figure 4.3, 4.4, 4.5 and 4.6 represent messages required during joining of nodes in TCP and CCP with different overlay size i.e. 32, 1024, 2048 and 4096. As stated earlier the value of τ is dependent on the size of the overlay. Figure 4.3-[A], 4.4-[A], 4.5-[A] and 4.6-[A] represent messages required for joining of bootstrapping nodes. During bootstrapping period a node joining in TCP behaves like node joining in CCP so messages required in both are same. The graphs shown here is the cummulative sum of each parameters in different time slots. For example, as shown in Figure 4.4-[C] the number of messages required in joining of nodes in time slot 20 is the cummulative sum of messages required in time slot [0–10] and [10–20].

The "*Upgrading Nodes in TCP*" is counted after bootstrapping period is over. Therefore, "*Upgrading Nodes in TCP*" is zero upto bootstrapping period which

is depicted in all graphs. After bootstrapping period a node joins in TCP needs upgrading messages. Initially when a node joins in TCP is known as temporary node which does not require any messages. The number of messages for *"Node Joins in TCP"* is calculated as the summation of messages required for *"Bootstrapping Nodes"* and "*Upgrading Nodes in TCP*".

It is observed that the joining of nodes in the system is not uniform. For example, as shown in Figure 4.3-[B] during time period [30–40] there is no joining of new nodes. So the graph of *"Node Joins in Chord"* during this period remains flat. But, in case of TCP there is upgradation of nodes and hence messages are required. In Figure 4.4-[B] from time stamp [20–30] there is no upgradation of nodes in TCP, but reflects joining of new nodes in CCP.

It is observed that to execute same number of joins in TCP and CCP requires different amount of messages. The *"Break Event Point"* (BEP) is defined as the time where TCP becomes same as CCP. It observed from Figure 4.4-[B], 4.5-[B], 4.6-[B] and 4.6-[C] if there is no leaving of nodes in the system then the number of messages required to join nodes becomes equal in both TCP and CCP i.e. in 4.5-[B] BEP is 68, in Figure 4.6-[B] BEP is 72 and so on.

The number of messages required for leaving of a node may be different in TCP and CCP. That is because of when a temporary node leaves the TCP system it incurred zero message pass. Also, the leaving of a quasi-stable node needs less messages in comparison to the node leaving in CCP.

As the event occurs randomly so number of updates may vary from time to time. But in all cases the number of messages required in TCP is always less than or equal to messages required in CCP.

From Table 4.3, it is observed that for same number of leave operations messages required is different in both systems. It is stated that leaving of a temporary node incurred zero message cost, leaving of a quasi stable node requires messages only upto the upgrading point and leaving of a fully-stable node requires same message pass as in CCP. The Table 4.3 represent some samples for network size 1024, where number of joins indicates join events out of respective numbers of events of column 1.

It is observed from the Table 4.4 that messages required for searching data in TCP and CCP is almost same which supports the design criteria of TCP.

Number of Events	Number of Joins	Number of Leave	Message in TCP	Message in Chord
100	37	1	0	28
150	56	9	76	110
200	53	10	100	110
250	79	10	85	109
300	90	10	61	87

Table 4.3: Message required to leave a node in TCP and Chord

Number of Events	Number of Joins	Number of Search	Message in TCP	Message in Chord
100	37	50	105	106
150	56	72	155	141
200	53	120	225	240
250	79	141	255	371
300	90	171	442	467

Table 4.4: Messages required to execute Point Query in TCP and Chord

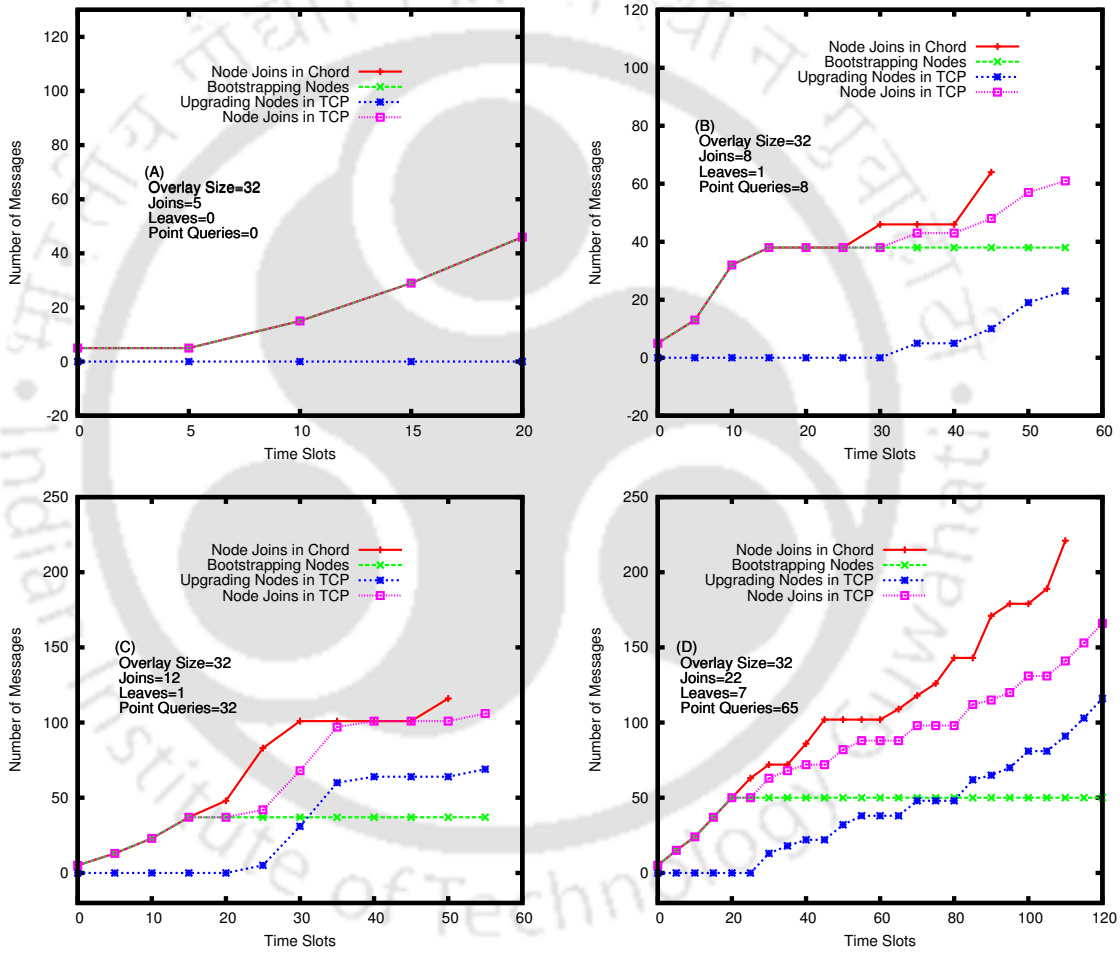

Figure 4.3: Graph showing number of messages required for Join and Upgrade of nodes in TCP and Chord (Overlay Size=32 and $\tau = 5$)

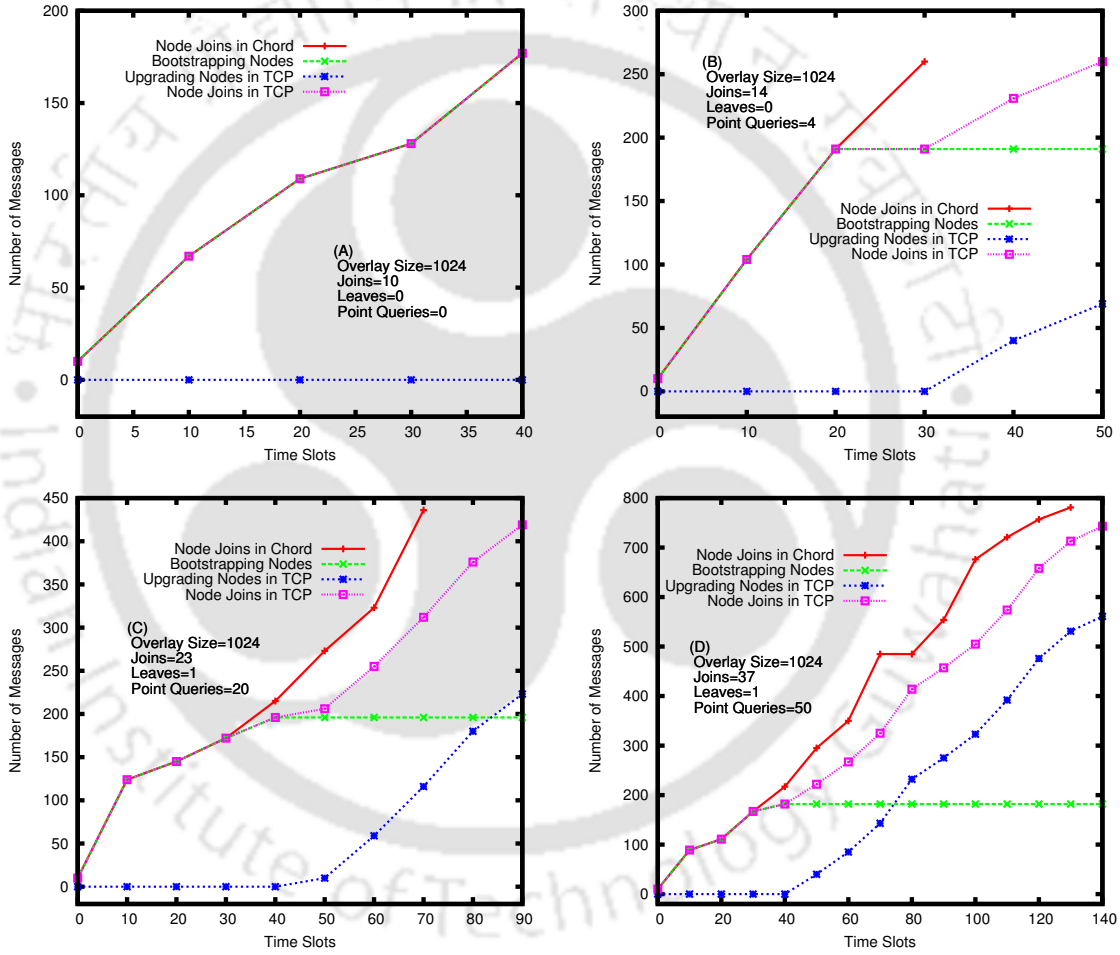

Figure 4.4: Graph showing number of messages required for Join and Upgrade of nodes in TCP and Chord (Overlay Size=1024 and $\tau = 10$)

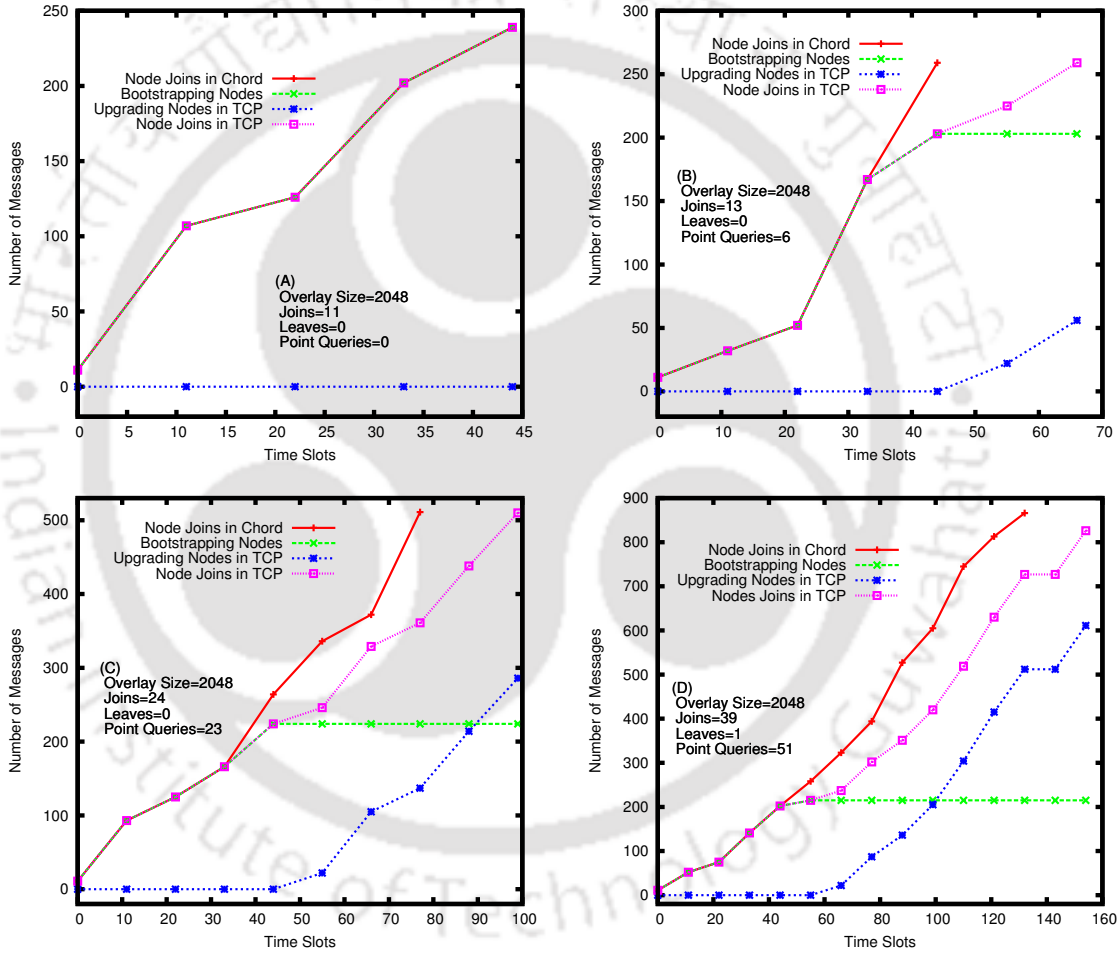

Figure 4.5: Graph showing number of messages required for Join and Upgrade of nodes in TCP and Chord (Overlay Size=2048 and $\tau = 11$)

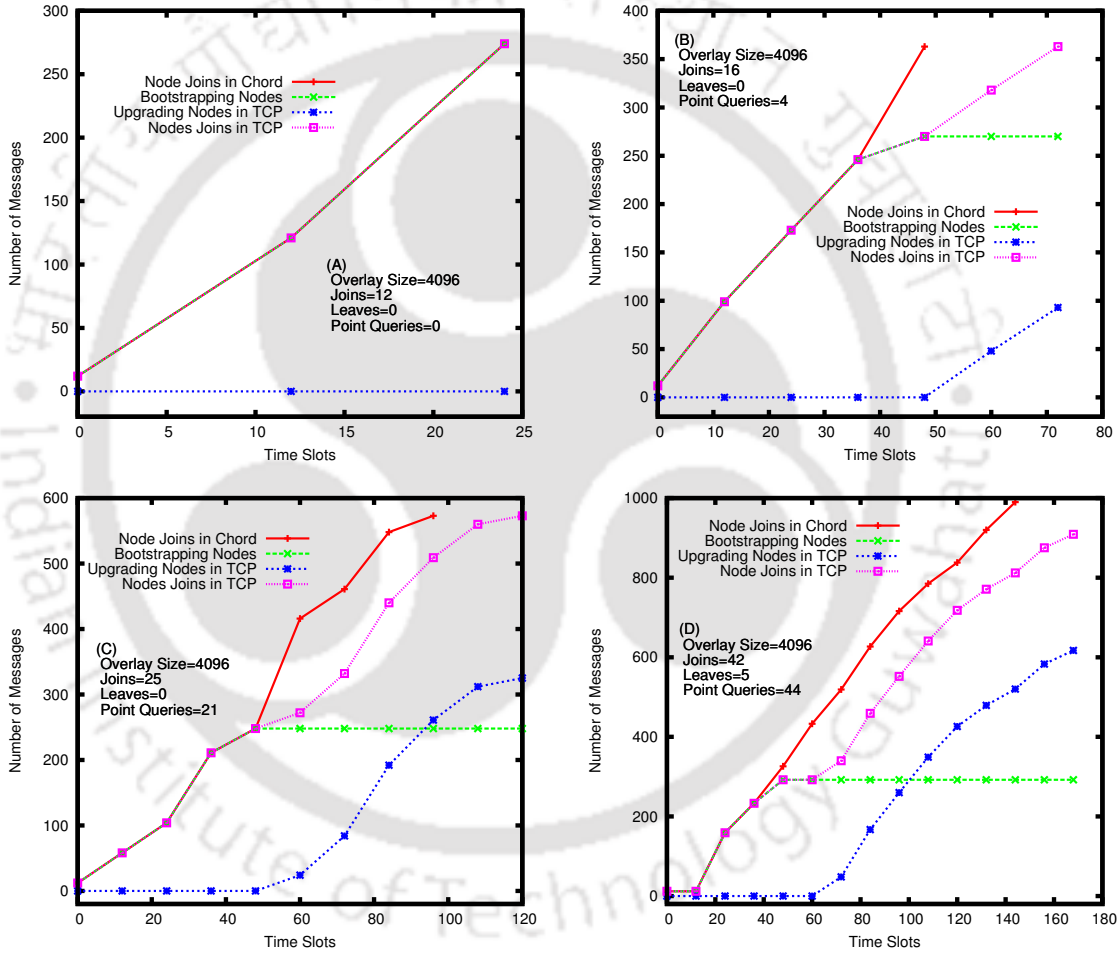

Figure 4.6: Graph showing number of messages required for Join and Upgrade of nodes in TCP and Chord (Overlay Size=4096 and $\tau = 12$)

4.4 Related Works

Chord is a distributed and deterministic lookup protocol for data sharing applications. Stoica et.al [SMK+01] describe the salient features of Chord protocol as simplicity, provable correctness and provable performance. There are many attempts have been done to improve the performance of Chord protocol, but some of them are mentioned here. Kaashoesk et al. [KK03] describe that if each peer maintains state for only two nodes, query processing is possible in $O(\log n)$. In the event of frequent joining/leaving of nodes this will also leads to performance degradation of the p2p system. Hong et al. [HLYW05] design PChord on the basis of Chord aiming to achieve better routing efficiency, which includes proximity routing in its routing algorithm. In PChord, next hop is not only decided by the entries in the finger list, but also decided by the entries in the proximity list. Yu et al. [SSJKQG07] describe a new DR-Chord (double-ring Chord), in which the resource is searched concurrently from two directions. Ye et al. [YPZJ05] propose a TCSChord which is based on the Topology-aware Clustering in Self-organizing mode. It can improve the efficiency of Chord routing. Here some rules are applied for a node to learn other nodes physical topology-aware locations. Xuan et al. [XCK03] construct RChord which is a modification to Chord system that is resilient to routing attacks. A concept of reverse edges is used. With the RChord system, performance efficiency is achieved that is greater than 50% under intensive attack conditions. Burresi et al. [BCRS08] propose MessChord which accounts for peculiar features of wireless mesh networks. MESHCHORD reduces message overhead of as much as 40% with respect to the basic Chord design. It improves the information retrieval performance. Li et al. [LY08] put a new optimized Chord algorithm that accelerates locating the hot resources and decreases the average query hops. Here each node is added with a key access counter and a hot key list. Wang et al. [WYG06] describe a bidirectional Chord system based on latency-sensitivity to decrease the latency and cost of locating resources. Kunzmann et al. [KBH05] analyze the ability of the Chord protocol

to keep the network structure up to date, even in environments with high churn rates. Here different designs have analyzed on the stability of the Chord structure. Jiang et al. [JY06] design a protocol, LLCHORD, which is used in routing query message to the destination. Here, while retain the $O(\log n)$ routing latency, the routing is aimed at reduce the cost of each hop. Here the cost of the each step to the destination node is reduced. The propose system enhances the performance of Chord protocol for more than 66.67% in average case without compromising the query processing.

4.5 Summary

In this chapter, a new approach called Temporal Centralization is introduced which improves the performance of Chord protocol by reducing the churn rate with a small overhead of maintaining an extra table called temporary routing table at each node. This model would be beneficial in an environment where the rate of joining and leaving of nodes is very high. Experimental results have been found to be encouraging.

Chapter 5

Unstructured Hierarchical Overlay Protocol

In an unstructured p2p architecture data is stored in nodes without any rules. Query routing is an expensive process in such unstructured systems. Some early systems like Gnutella used pure flooding for query routing which functions like Breadth First Search on a graph with depth limit D. The parameter D is the system-wide maximum time-to-live (TTL) of a message in terms of overlay hops. The querying node sends a request to all its neighbors. Each neighbor processes the query and returns the result if the data is found. If a neighbor receiving the request is not able to provide the requested resource, it forwards the query further to all its neighbors except the querying node. This procedure continues until the depth limit D is reached. Flooding tries to find the maximum number of results within the ring that is centered at the querying node and has the radius of D-overlay hops. However, it generates a large number of messages (many of them are duplicate messages) and does not scale well. One later version of Gnutella based on scoped-flooding to limit the problem of looping in original flooding. With scoped-flooding, each message is flooded only to the nodes within a given fixed distance from the source. The distance is given by a TTL flag that defines the broadcasting scope.

Unstructured overlays have the disadvantage that queries may take a long time to find required data or there is no guarantee to locate an existing data item. Flooding enables communication without routing table or knowledge of the specific network topology. Unrestricted flooding requires tremendous number of message transmissions and network resources, and does not scale well. In this report, we present an organized network architecture for unstructured peer-to-peer systems where nodes are added to the network in a systematic way to efficiently utilize the resources of the nodes in the network. This network architecture is characterized by $O(\log_m n)$ network diameter and $O(\log_m n)$ messages for node joining and node failure, where n is the number of nodes in the network and m is the maximum number of children of a node. Purely decentralized systems like Gnutella route the query in an environment where the node capabilities are not identified. Whereas the proposed unstructured hierarchical overlay protocol (HUP) routes query towards the *high capable nodes*. This organization of nodes improves the query success rate compared to that of purely unstructured systems.

We first define different terms used in the subsequent sections in section 5.1 and give the system architecture in section 5.2. Algorithms used for various operations are described in section 5.3. Experimental results are discussed in section 5.4 and section 5.5 discusses the related works for this chapter. Section 5.6 presents the summary of the chapter.

5.1 Terminologies Used

i. ***Complete node***: A node in the network is said to be complete, if it has m child nodes, where m is the maximum number of children a node can have.

ii. ***High capability node***: A node P is said to be a high capability node in comparison to node Q, if it provides more sharing to the network than

that of Q and is more stable than Q. (*Stability of a node is defined by the amount of time for which it is in the p2p network. More time a node is in the network, more is the stability*). Otherwise node P is said to be a low capability node in comparison to node Q.

iii. ***Token***: This is a control frame that moves throughout the cluster. Only the Token holder is allowed to add nodes to the cluster. Token flow in the network controls the network growth and ensures hierarchy of nodes in the system.

5.2 HUP Structure

Nodes in a network differ in terms of their computing power, communication capacity, stability, available memory and sharing size. In our approach of unstructured hierarchical overlay protocol (HUP), a node's position in the network is determined by it's capabilities (sharing size). The nodes are categorized as: *super node* and *normal node*. Super nodes are placed in the first level of hierarchy. These are assumed to be in the network for most of the time and are connected using mesh topology. Every super node has a *Token* to build the hierarchy of nodes in its cluster. All super node *ids* are maintained in the super node table. Table 5.1 shows the super node table for super node S_0 of Figure 5.1. The super node table keeps records of other super nodes along with their time of initiation in the system. The table is used during inter-cluster communication.

Super node ID	Time
S_1	t_1
S_2	t_2
S_3	t_3

Table 5.1: Super Node Table of S_0

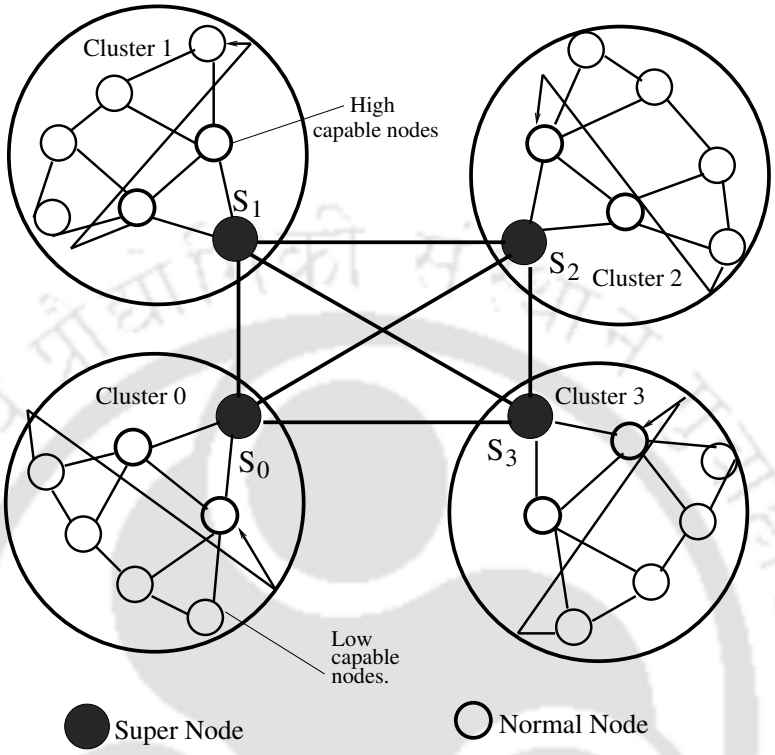

Figure 5.1: Illustration of High-level Architecture of HUP

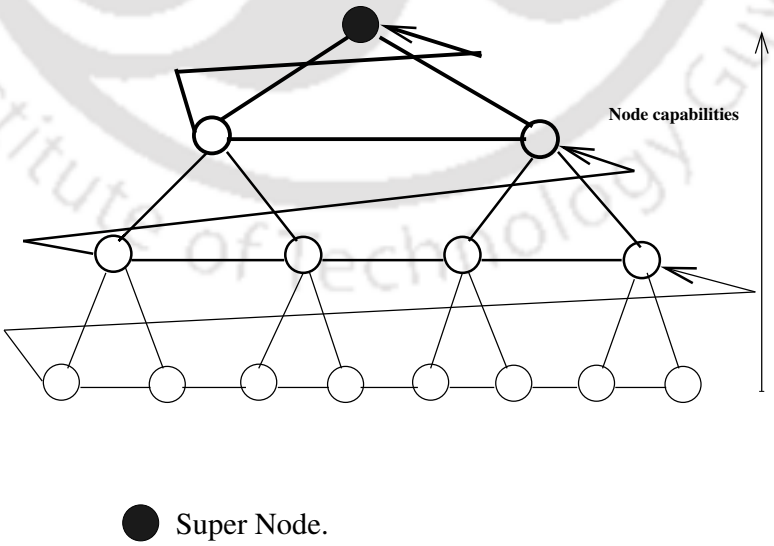

Figure 5.2: Illustration of a Cluster in HUP

Figure 5.1 shows the hierarchical structure with four clusters and $m = 2$, where m is the maximum number of children a node can have. Number of nodes in a cluster may vary from one cluster to another. For instance, some clusters may have thousands of nodes while others may have only few. Figure 5.2 illustrates the structure of a cluster of HUP as shown in Figure 5.1. The connected nodes have information of their siblings, children and parents. The network grows adding nodes from left to right. After completing a level new nodes are added to the network from leftmost side. Nodes in a cluster are differentiated by the amount of sharing, bandwidth and processing power etc. Nodes with relatively high sharing, which are called as high capability nodes are placed in the upper levels of the hierarchy and low capability nodes are placed in the lower levels of the hierarchy.

5.2.1 Bootstrapping

At the startup, nodes that join the network are treated as super nodes. These nodes are connected by mesh network. Each super node has a Token to build the hierarchy in its cluster and it is assumed that it stays in the network for considerably long time. Other nodes join the network by connecting to super nodes or normal nodes.

A node that wants to join the network searches for already connected nodes in the network. If any of the nodes are found, procedures given in section 5.3 are used for establishing the connections. If it does not find any of the nodes already present in the network in its proximity, it contacts one of the super nodes to join the network. Every node in the system has the information of at least one of the super nodes.

Each node (super and normal) maintains two routing tables - *active routing table* and *passive routing table* - for routing queries and to deal with node failures, respectively. The format of active routing table and passive routing table

are shown in Table 5.2 and Table 5.3, respectively. Active routing table maintains information of immediate neighbors whereas passive routing table maintains information of neighbors one hop away. Identity, sharing size and time stamp of the node in the network are maintained in the above mentioned table. As the nodes change their sharing capacity, their position in the hierarchy changes and accordingly the table is updated.

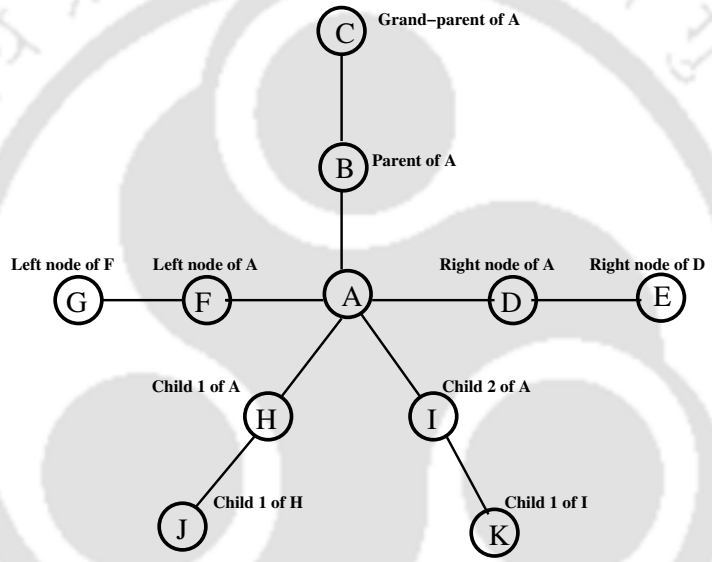

Figure 5.3: Connection of a node A

Figure 5.3 shows the connections of a sample node, A, in HUP. Tables 5.2 and 5.3 shows the active routing table and passive routing table for node A.

Active routing table maintains the information of the immediate neighbor. As shown in Table 5.2 for node A the immediate parent is node B, immediate left node is F, immediate right node is D and the immediate child nodes are H and I, respectively.

The passive routing table of node A is shown in Table 5.3. Information regarding neighbors one hop away from the node A - specifically the Grandparent

Type	NodeID	sharing	Time
Parent	B	1000	t_1
Sibling1	F	500	t_1
Sibling2	D	500	t_1
Child1	H	400	t_1
Child2	I	400	t_1

Table 5.2: Active Routing table of node A.

C, left-left node G, right-right node E, left-left child J and right-right child K - are kept in this table.

Type	NodeID	sharing	Time
Grandparent	C	2000	t_1
Sibling11	G	500	t_1
Sibling21	E	500	t_1
Child11	J	200	t_1
Chid21	K	200	t_1

Table 5.3: Passive Routing table of node A

5.3 Algorithms

This section provides the basic algorithms used in HUP overlay system.

1. **Handle-Token**: This algorithm describes how a node handles the Token. Token moves from one node to other in the hierarchy to ensure that the network grows in an ordered fashion. The steps of the algorithm are given below.

- Step-1: Whenever a node which is not a complete node receives the Token from its sibling it tries to add node to itself to become a complete node and hence keeps the Token.
- Step-2: Otherwise, if it is already a complete node, it passes the Token to its right sibling.
- Step-3: If there is no right sibling, then the Token is passed to the parent node. This is continued till the parent is null. If the parent is null, the Token is passed to the first left child. This is continued till the leftmost node of the tree is found.

2. **Find-Token**: This is run by a node whenever a new node contacts it for joining the network or when some node sends "Find-Token" message. Whenever a Token is lost in the system due to high churn rate or any other reason then the super node generates a new Token. Let a new node P has contacted the node Q for joining the network. Steps of the algorithm are as given below.

   - Step-1: If Q has the Token, P is directly added to the network using algorithm "Add-Node" (described after this).
   - Step-2: If Q does not have the Token, it extracts the *id* of the node, say M, from the log (log of a node contains the *id* of the nodes to which it has passed the Token most recently) to which it has passed the Token and passes the request to that node with its *id* as the source *id*. Here, source *id* means the *id* of the node which is initiating the request.
   - Step-2.1: If the node M to which the request was sent by Q does not have the Token, then M repeats step-2 till the Token is found and the node *id* is reported to the source node.
   - Step-3: If there are no entries in the log, the node forwards the request to the parent node with it's *id* as the source node. This is continued

till the parent becomes null (i.e. request arrived at super node). The super node creates a new Token and pass it to the parent of the last node of the cluster.

3. **Add-Node**: This algorithm is run by a node when it wants to add a new node to the group. Let an existing node O wants to add a new node P in the group. $P.S$ and $O.S$ are assumed to be sharing size of P and O, respectively. The steps of the algorithm Add-Node as given below.

   - Step-1: If $O.S > P.S$ and O is not complete then simply add P as a child node of O.
   - Step-2: Increment the child count of node O.
   - Step-3: *Handle Token (node O)*
   - Step-4: If $O.S < P.S$, add the node to O and call *Balance-Network*(O, P)
   - Step-5: Update active and passive routing tables of nodes.

   Based on the sharing size of the new node, it is placed in the appropriate level in the hierarchy. Algorithm Balance-Network given below describes the way the hierarchy is balanced if a new node P replaces a node Q already in the system.

4. **Balance-Network**(O, P): It is called whenever the sharing size of the existing node, to which the new node contacts to join, is less than the sharing size of the new node. The steps are as given below.

   - Step-1: If $O.S < P.S$, replace node O with node P.
   - Step-2: If node O holds the Token, it is passed to node P.
   - Step-3: Increment the child count of node P.
   - Step-4: If $P.S < Parent(P).S$ then stop. Otherwise, *Balance-Network*$(Parent(P), P)$, where $Parent(P)$ is parent of node P.

- Step-5: Update routing tables for nodes.

5. **Remove-Node**: This algorithm is executed whenever a node leaves the network and there is a requirement of balancing the whole network. The steps are as given below.

   - Step-1: If the leaving node is a leaf node, decrement the child count of its parent node.
   - Step-2: If it holds the Token, it is passed to the right sibling.
   - Step-3: If it is the rightmost leaf node then it is passed to the parent until the parent is null and then it is passed to the leftmost child of the network.
   - Step-4: If the leaving node is not leaf node then lift its leftmost child as the parent node and call *Balance-Network(*P, C*)*, where P is the parent of the leaving node and C is the leftmost child of the leaving node.
   - Step-5: Update the routing tables of nodes.

5.3.1 Node Joining

Based on the above algorithms, node joining process in HUP is described as follows

i. When a node wants to join the network, it sends k ping messages to already connected nodes in the network.

ii. Based on the observed Round Trip Time (RTT), new-node sends the "Join" request to the node which is within its proximity.

iii. The receiving node uses algorithm Find-Token to find the *id* of the current Token holder and gives it to the new node.

iv. The new node sends the message to the Token holder.

v. Token holder executes the algorithm Add-Node to add the node to the system.

As explained in algorithm Handle-Token, the Token of the network moves from one level to other in an ordered fashion. Token movement ensures that the network grows level by level. In HUP, number of children of a node is fixed to m. With n nodes in the cluster and m being the maximum number of children of each node, the height of the network is $O(\log_m n)$. In worst case, the Token holder is at the leftmost bottom of the hierarchy and a new node contacts the node at rightmost bottom of the hierarchy. In the worst case "Find-Token" message will be transferred from each node to its parent node and then from the topmost node it will again be transferred to its leftmost child node. Since the height of the network is $O(\log_m n)$ it will completely traverse the height of the network twice. Thus, the total number of messages transferred is $2\log_m n$ in the worst case. The algorithm Add-Node adds the node to the corresponding hierarchy based on its capabilities. This in turn uses algorithm Balance-Network which balances the network along the path of insertion. When a node is added to the network, at most $4\log_m n$ messages are passed in the overlay network. Thus, addition of a node takes $O(\log_m n)$ messages.

5.3.2 Query Processing

Initially a node forwards the query towards the upper levels of the network. If the node can not find the data with maximum allowable TTL, it forwards the query towards the lower levels of the network. For the first alternative, source node forwards the query to parent, left and right nodes. For the second alternative, source node forwards the query to left child, left and right nodes. Then each node forwards the query to it's left child. Algorithm Query-Up and Query-Down describe the way a query is processed in the further steps.

Algorithm 6: Query-Up

Input: Query Q, Source node S, Time-to-Live value TTL

- Step-1: The query Q is initiated by the node S.
- Step-2: If TTL is greater than 0 go to step-3, otherwise stop.
- Step-3: If the query Q matches the local index of the node, then sends response.
- Step-4: If S is a super node, then the query is forwarded to the left child of S and broadcast to super nodes of other clusters. S then executes the algorithim Query-Down.
- Step-5: Otherwise, the query is forwarded to the left node, right node and the parent node.
- Step-6: Decrement TTL value and go to Step-2.

Algorithm 7: Query-Down

Input: Query Q, Source node S, Time-to-Live value TTL

- Step-1: The query Q is initiated by by the node S.
- Step-2: If TTL is greater than 0 go to step-3, otherwise stop.
- Step-3: If the query Q matches the local index of the node, then sends response.
- Step-4: If S is a super node, then the query is forwarded to the left child of S.
- Step-5: Otherwise, the query is forwarded to the left node, right node and left child node.
- Step-6: Decrement TTL value and go to Step-2.

In order to decrease the number of unnecessary messages and increase the probability of finding the desired data soon, initially the query is forwarded to the upper levels. If a super node gets the query, it forwards the query to all the super nodes which in-turn executes the algorithm Query-Down. But in the neighboring clusters the query moves down the hierarchy, giving priority to the high capable nodes. If the querying node does not get the desired response then the query is forwarded to the lower levels of the network.

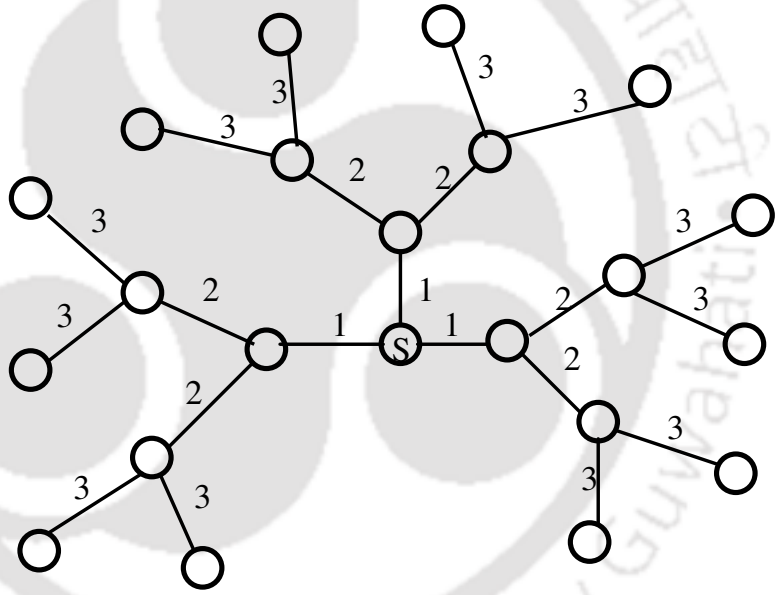

Figure 5.4: Routing in Gnutella for degree 3, label indicates TTL value.

Figure 5.4 shows the way a query is forwarded when modeling the Gnutella as a random graph. In HUP, as shown in Figure 5.5, a query is initially forwarded toward more capable nodes, whereas in Gnutella the query is forwarded to unidentified nodes. Let N_{hup} and N_{pure} be the number of nodes receiving query in HUP and pure unstructured (e.g. Gnutella), respectively. We observe that

$$\text{for } TTL = 0, \qquad N_{hup} = 1;$$
$$\text{for } TTL = 1, \qquad N_{hup} = 1 + 3; \textit{and}$$

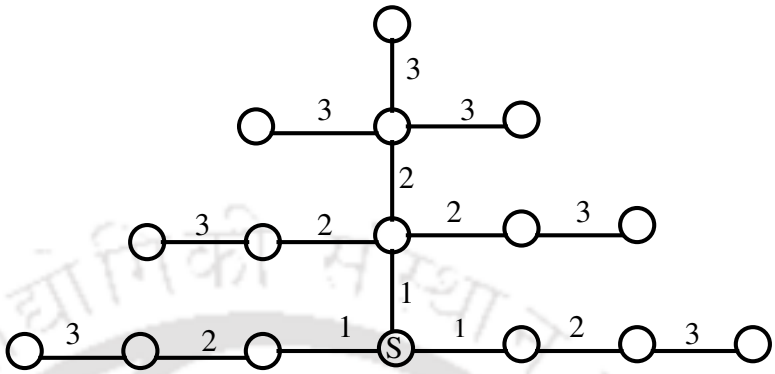

Figure 5.5: Query routing in HUP, label indicates TTL value

for $TTL = 2$, $N_{hup} = 1 + 3 + 5$.

In general, for $TTL = K$

$N_{hup} = \sum_{i=0}^{K}(1 + 2 \times i)$

$= \frac{2\times(K+1)\times(K+1)}{2}$

$= (K + 1)^2$

Let there be P super nodes in the system. If a super node queries, then the number of nodes which receive the query gets multiplied by P.

Hence, $N_{hup} = P \times (K + 1)^2$.

If the average TTL be K_{avg}, then the number of nodes in HUP is

$N_{hup} = P \times (K_{avg} + 1)^2$ (1).

Let M be the maximum degree of a node in the network in pure unstructured system. Then we see that

for $TTL = 0$, $N_{pure} = 1$;

for $TTL = 1$, $N_{pure} = 1 + M$;

for $TTL = 2$, $N_{pure} = 1 + M + M \times (M - 1)$;

for $TTL = 3$,

$$N_{pure} = 1 + M + M \times (M-1) + M \times (M-1)^2.$$

In general, for $TTL = K$

$$\begin{aligned} N_{pure} &= 1 + M \sum_{i=0}^{K-1} (M-1)^i \\ &= \frac{M \times (M-1)^K - 2}{M-2} \\ &\approx (M-1)^K \end{aligned} \quad (2)$$

From the Equation (1) and (2), it is observed that N_{hup} increases polynomially and N_{pure} increases exponentially. Hence, pure unstructured system routes the query towards the non-deterministic nodes and the HUP routes the query towards the probable nodes in the network. Therefore, bandwidth consumption is exponential in purely decentralized architecture, whereas in the proposed architecture it is polynomial.

We assume uniform distribution of data across the nodes in the network. Let p be the probability of a node providing the desired data. Probability p is a function of node capabilities, node availability and link state. Figure 5.6 shows the probability distributions in HUP architecture. Nodes in the same level are given equal probability, because nodes in the same level are approximately equally capable. Hence, nodes in the upper levels have higher probability i.e. $p2 > p1 > p$.

Let P_{hup} be the probability of finding data in HUP system and probability of finding data at the source node is p. Then the probability of finding data in the directly connected neighbors is
$P_{hup} = 1 - (1-p)^3(1-p3)^2(1-p1)$, where $p3$ is the probability of finding data in the child nodes of node S (note that $p2 > p1 > p > p3$).

Now, the probability of finding data in two hops is given as
$P_{hup} = 1 - (1-p)^5(1-p3)^6(1-p4)^4(1-p1)^3(1-p2)$.

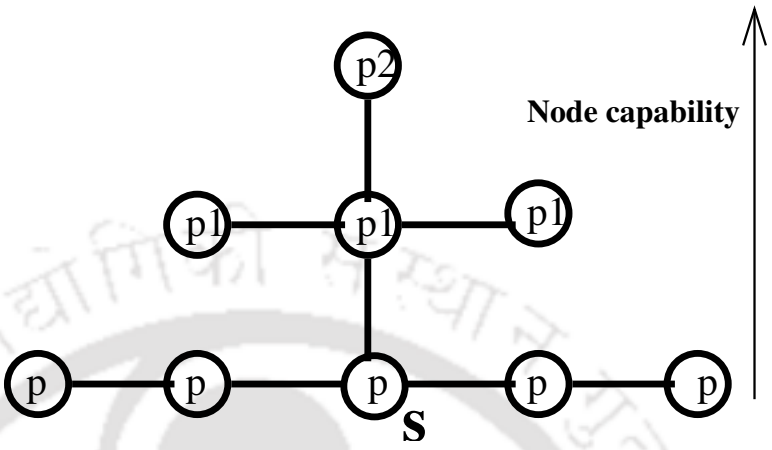

Figure 5.6: Proability distribution in HUP, where $p2 > p1 > p$ and S is the source node.

Figure 5.7 shows the probability distribution in pure Gnutella. Let a be the average probability of finding data in the system.

Let P_{pure} be the probability of finding data in Gnutella. Then the probability of finding data in directly connected nodes is

$P_{pure} = 1 - (1 - a)^{M+1}$

Let the degree of gnutella be 4 i.e. $M = 4$. Then,

$P_{pure} = 1 - (1 - a)^5$

Now the probability of finding the data with in two hops is

$P_{pure} = 1 - (1 - a)^{M+1}(1 - a)^{M(M-1)}$.

If single hop is considered for both HUP and pure unstructured system, then the ratio is found as

$\frac{P_{hup}}{P_{pure}} = \frac{1-(1-p)^3(1-p3)^2(1-p1)}{1-(1-a)^5}$

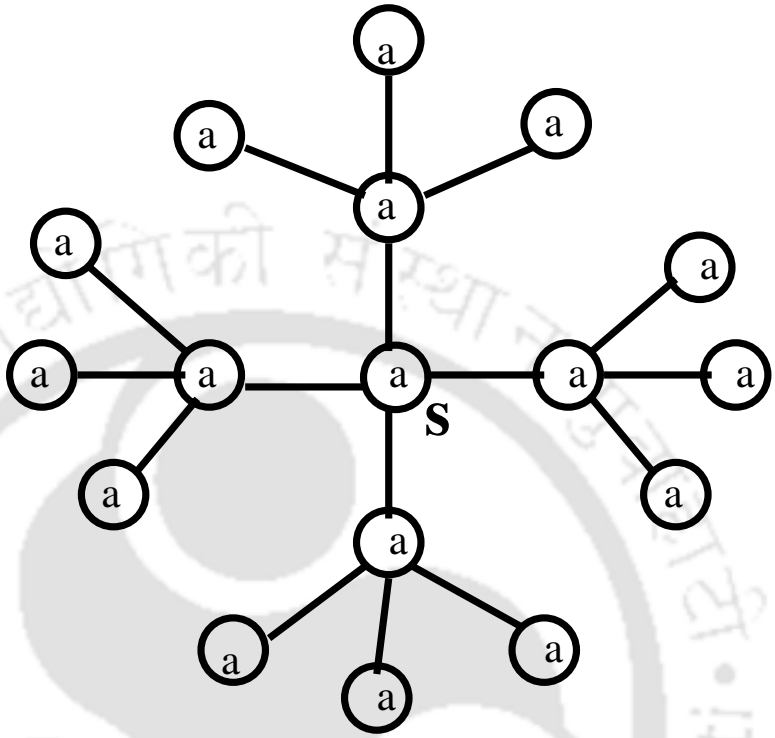

Figure 5.7: Probability distribution in Gnutella with degree 4 and probability a.

Putting $a = p$

$$\frac{P_{hup}}{P_{pure}} = \frac{1-(1-p)^3(1-p3)^2(1-p1)}{1-(1-p)^5}$$

Assuming the value of P_{hup} to be greater than P_{pure}. We find that

$$1-(1-p)^3(1-p3)^2(1-p1) > 1-(1-p)^5$$
$$\Rightarrow (1-p)^2 > (1-p3)^2(1-p1)$$
$$\Rightarrow p^2 + 2p3 + p1 + p1p3^2 - 2p - p3^2 - 2p3p1 > 0$$
$$\Rightarrow (p-1)^2 - 1 + 2p3 - p3^2 + p1 + p1p3^2 - 2p3p1 > 0$$
$$\Rightarrow (p-1)^2 - (p3-1)^2 + p1(p3-1)^2 > 0$$
$$\Rightarrow (p-1)^2 + (p3-1)^2[p1-1] > 0$$

Now,

- $(p3-1)^2[p1-1] < 0$ since $(p1-1) < 0$.
- $(p-1)^2 > (p3-1)^2$ since $p > p3$.
- $(p1-1)$ and $(p3-1)^2$ are less than 1.

So, $(p3-1)^2[p1-1] < (p3-1)^2$, which proves the satisfactory condition for the inequality
$(p-1)^2 + (p3-1)^2[p1-1] > 0$.

Therefore, $P_{hup} > P_{pure}$

Hence, HUP improves the probability of finding the data when compared to pure unstructured system.

5.3.3 Edge Congestion

In HUP, each node initially forwards the query Q to the upper levels of the network. Greedy nodes can forward more queries into the network and make the nodes in the upper levels heavily loaded, which affects the system performance. This problem can be avoided by the following mechanism of dividing the upload bandwidth among the directly connected nodes.

Upload Bandwidth (B_G) of node G as shown in Figure 5.8, is shared among the connected nodes.

Let, S_A, S_C, S_D, S_E and S_F be sharing sizes of A, C, D, E and F, respectively. Let B_A, B_C, B_D, B_E and B_F represent the proportion of bandwidth of node G allocated to nodes A, C, D, E and F, respectively.
Let, $sum = S_A + S_C + S_D + S_E + S_F$

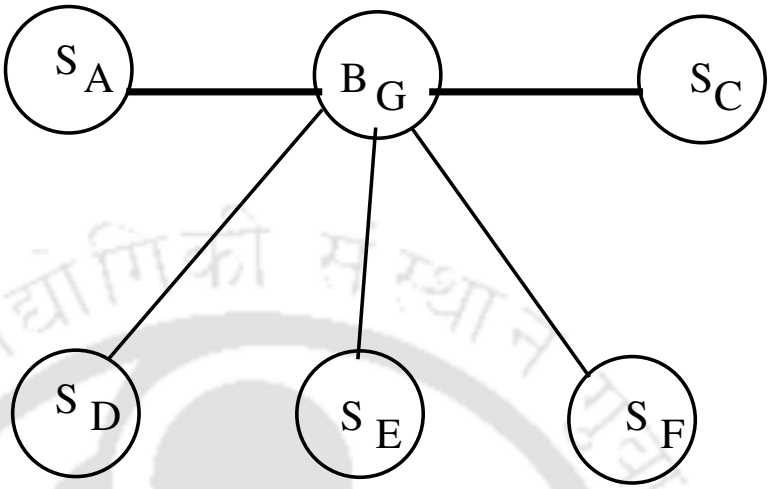

Figure 5.8: Illustration of bandwidth allocation

Hence,

$B_A = \frac{S_A}{sum}$

$B_C = \frac{S_C}{sum}$

$B_D = \frac{S_D}{sum}$

$B_E = \frac{S_E}{sum}$

$B_F = \frac{S_F}{sum}$

This strategy regulates the free-riders, which are greedy to consume the resources and leave the network very soon. Since each node regulates the rate at which the queries are processed, the flow of queries is regulated and free-riders are controlled.

5.4 Experimental Results

We have implemented one cluster of HUP taking maximum number of children as 2 (i.e. $m = 2$). Experiments results are taken with varying network sizes. Input data is generated according to the Table 5.6 which indicates an analysis [ZyF02] of Gnutella. Each of the file type is mapped on to the number space according to their percentage of queries. For example, if total data space size be 100, then 19 of key *id*s be assigned for *.avi* files (which fall in the rage of 1 to 1000) as shown in Table 5.6.

5.4.1 Node Joining

The system needs balancing according to the sharing capacity of incoming nodes. Table 5.4 shows the number of nodes joining the network and corresponding total messages required to balance the network. Figure 5.9 represents the corresponding graph. As described in the section 5.3.1, in worst case, the number of messages for a node to join in the system is $O(\log_m n)$. It is observed that the number of balancing messages required is always less than $O(\log_2 n)$.

5.4.2 Query Processing

Table 5.5 shows the top 20 queries on Gnutella network. Table 5.6 shows the top 20 file types requested on Gnutella network with mapping on to the number space for simulation.

Table 5.7 shows the success rate with increase of TTL value. Figure 5.10 and 5.11 shows the query success rate vs TTL of the query. As the TTL of the query increases the success rate of queries increases. For smaller networks, the success rate for small TTL may be high. Figure 5.12 and 5.13 shows the success rate vs network size. Observations reveal that with $TTL = \log_m n$, success rate is at least 50% in HUP which is an improvement over pure unstructured system like Gnutella where only 7 to 10% queries are successful [ZCSK07].

Number of nodes added	Total number of messages	Average no of messages per join
16	14	0.88
32	27	0.84
64	59	0.92
128	144	1.13
256	283	1.11
512	594	1.16
1000	1121	1.12
2000	2483	1.24

Table 5.4: Number of nodes added and total message required

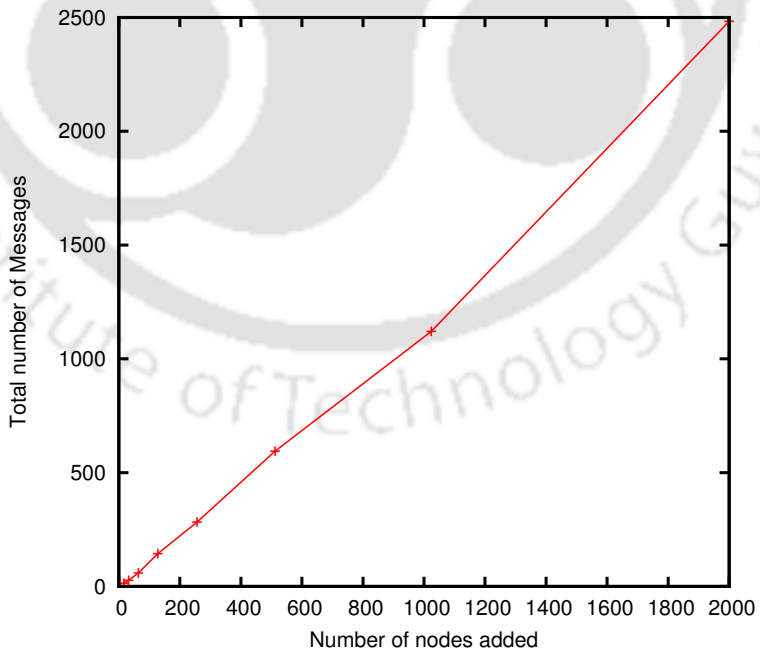

Figure 5.9: Messages required for addition of nodes

#	Query	%
1	divx avi	3.88
2	spiderman avi	0.33
3	p_mpg	0.25
4	star wars avi	0.25
5	avi	0.19
6	s_mpg	0.18
7	Eminem	0.18
8	eminem mp3	0.16
9	dvd avi	0.16
10	b_	0.16
11	divx	0.16
12	spiderman	0.15
13	xxx avi	0.14
14	capture the light	0.14
15	buffy mpg	0.13
16	g_mpg	0.13
17	buffy avi	0.13
18	t_mpg	0.12
19	seinfeld vivid	0.12
20	xxx mpg	0.12

Table 5.5: Top 20 Queries on Gnutella

Edge Congestion

Table 5.8 shows the bandwidth factor to access a node in HUP. The results shown implies that a node with a lower sharing size has lower bandwidth factor. It is observed that a free-rider node gets lowest bandwidth factor.

#	Filetype	%	Mapped Number space
1	avi	18.72	1 – 1000
2	mp3	17.84	1001 – 2000
3	mpg	13.10	2001 – 3000
4	ra	8.5	3001 – 4000
5	rm	2.79	4001 – 5000
6	zip	2.64	5001 – 6000
7	mpeg	2.63	6001 – 7000
8	jpg	1.9	7001 – 8000
9	asf	1.11	8001 – 9000
10	ps	0.97	9001 – 10000
11	mov	0.95	10001 – 11000
12	pdf	0.51	11001 – 12000
13	rar	0.44	12001 – 13000
14	exe	0.4	13001 – 14000
15	wav	0.25	14001 – 15000
16	doc	0.21	15001 – 16000
17	txt	0.07	16001 – 17000
18	gz	0.07	17001 – 18000
19	html	0.02	18001 – 19000
20	jpeg	0.02	19001 – 20000

Table 5.6: Top 20 File types requested on Gnutella in queries with mapping of file types on to number space.

5.5 Related work

Recently, different approaches have been proposed for structured peer-to-peer system based on hierarchical DHT (Distributed Hash Table). Li et al. [LHLH03]

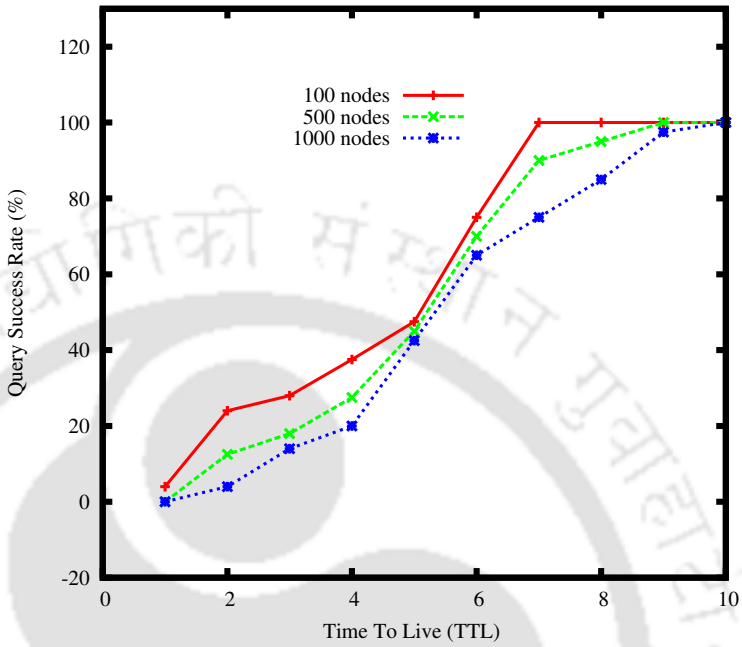

Figure 5.10: Query Success Rate vs TTL for network size 100, 500 and 1000

suggest one hierarchical aggregation network by introducing the concept of supernode. They introduce meta data files and constructed hierarchical architecture based on various content of sharing resources. Liu et al. [LLZ07] present a p2p design that employs the superpeer concept and explores the balance of lookup latency, transmitted messages and network expandability. Hudzia et al. [HKO05] describe a tree based peer-to peer network that constructs a tree based on leader election algorithm. The TreeP topology efficiently uses the heterogeneous aspect of the network while limiting the overhead introduced by the overlay maintenance. A multi-tier capacity aware topology is presented in [SGL04] to balance the load across the nodes so that low capable nodes do not downgrade the performance of the system. They propose capacity-based techniques to structure the overlay topology with the aim of improving the overall utilization and performance by developing a number of system-level facilities. They propose a overlay network scheme with Multi-Tier Capacity Aware Topologies, capacity aiming at

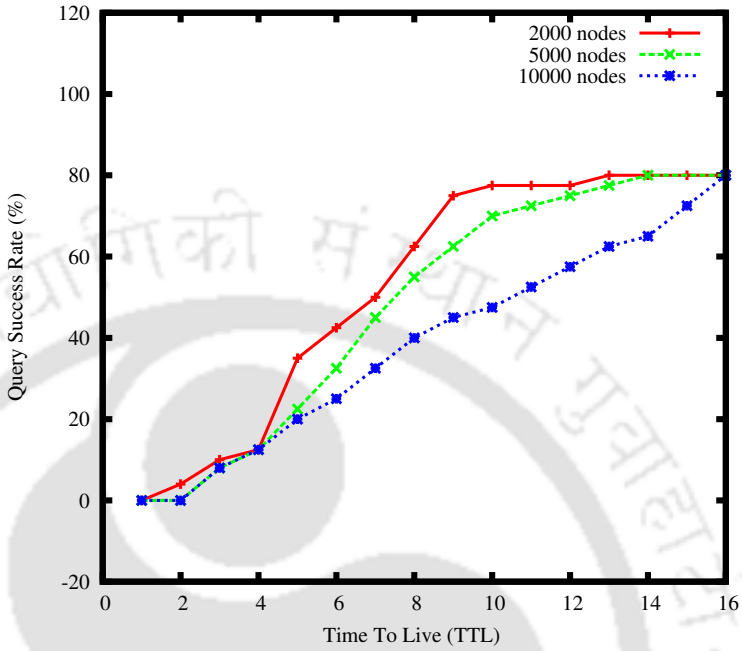

Figure 5.11: Query Success Rate vs TTL for network size 2000, 5000 and 10000

improving the network utilization, reducing the search latency, and improving the fault tolerance of the p2p system. They also developed an analytical model that enables us to construct multi-tier network-connection aware topologies such that the number of lookup queries served by a peer is commensurate with its capacity. Min et al. [CCF04] describe a super peer based framework in which the peers are organized based on their responsibilities. Normal peers always send the query to the super peers. They suggest the following three approaches: First, measure the capacities related to distance cost and the prediction based processing power between peers. Second, analyze common features indicating the similarity of content of desired files and the similarity of behavior between peers. Finally, in order to give a ranking to super peer, they compute maximum capacity and maximum similarity. The main contribution is reduction of the bandwidth cost by selection of best super peer. Terpstra et al. [TKLB07] propose a simple probabilistic search system, BubbleStorm, built on random multi-graphs. Their

TTL	Number Of Nodes					
−−	100	500	1000	2000	5000	10000
1	4	0	0	0	0	0
2	24	12.5	4	4	0	0
3	28	18	14	10	8	8
4	37.5	27.5	20	12.5	12.5	12.5
5	47.5	45	42.5	35	22.5	20
6	75	70	65	42.5	32.5	25
7	100	90	75	50	45	32.5
8	100	95	85	62.5	55	40
9	100	100	97.5	75	62.5	45
10	100	100	100	77.5	70	47.5
11	100	100	100	77.5	72.5	52.5
12	100	100	100	77.5	75	57.5
13	100	100	100	80	77.5	62.5
14	100	100	100	80	80	65
15	100	100	100	80	80	72.5
16	100	100	100	80	80	80

Table 5.7: Query Success Rate(%)

	Parent	Left Sibling	Right Sibling	Left Child	Right Child
Sharing Size	40	31	39	1	4
Bandwidth Factor	0.3478	0.2696	0.3391	0.0087	0.0348

Table 5.8: Bandwidth factors with reference to a node with sharing capacity 22

primary contribution is a flexible and reliable strategy for performing exhaustive search. BubbleStorm also exploits the heterogeneous bandwidth of peers. However, it sacrifices some of it's bandwidth for high parallelism and low latency.

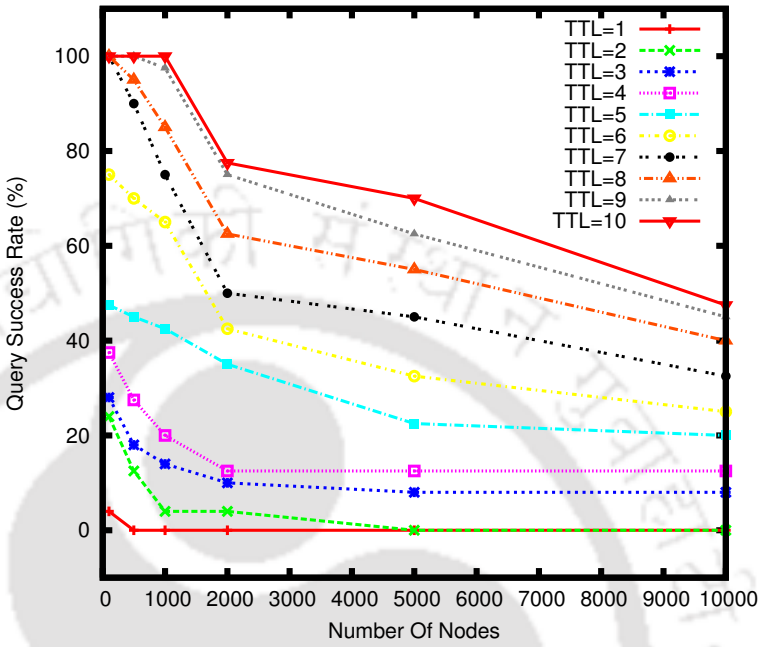

Figure 5.12: Query Success Rate vs Network Size for TTL value 1 to 10

The provided search guarantees are tunable, with success probability adjustable well into the realm of reliable systems. In unstructured p2p networks, effective search strategies are developed based on constrained flooding and random walks. But in case of Bubblestorm a data replication scheme is employed which does not take query distribution into consideration, potentially leading to load-balancing problems. Condie et al. [CKGM04] suggested one adaptive p2p protocol based on the fundamental notions as peers should directly connect to those peers from which they are likely to download satisfactory files and peers may use past history to determine the peers from which they are likely to download satisfactory files. Peers connect to those peers that have high scores, and disconnect from peers with low scores. In our approach of HUP, a hierarchical overlay network structure is designed based on the sharing size and stability of nodes. A node gets access to the resources based on its contribution level. This approach of routing the queries regulates free riders, which consume the system resources.

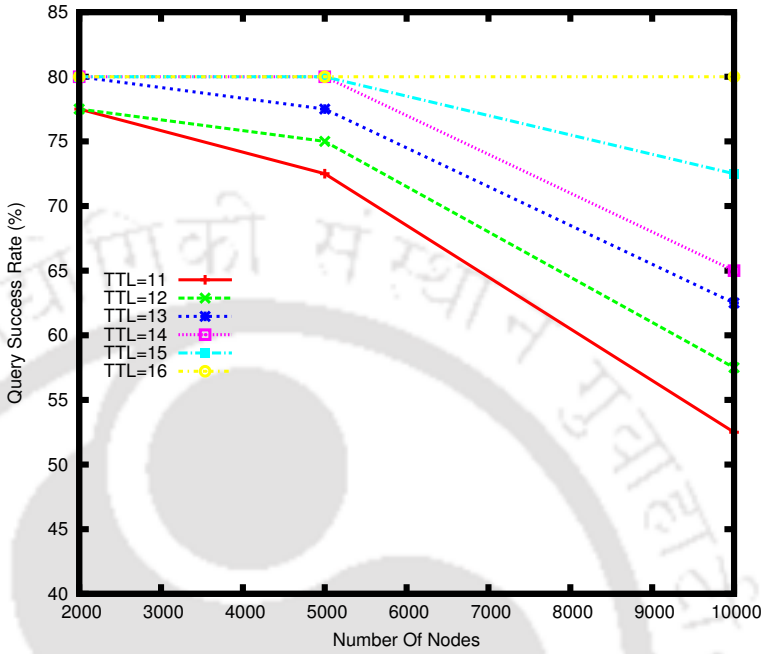

Figure 5.13: Query Success Rate vs Network Size for TTL value 11 to 16

	Gnutella	HUP	Chord
Diameter	n	$log_m n$	$log_2 n$
Size of routing table	Not fixed	$m+3$	$log_2 n$
Join(Messages)	Varies	$O(log_m n)$	$O(log_2 n)^2$
Deletion(messages)	Varies	$O(log_m n)$	$O(log_2 n)^2$
Node Differentiation	No	Yes	No

Table 5.9: Comparison of different systems

5.6 Summary

Due to unorganized structure query processing in unstructured overlays found to be very inefficient. In this chapter, an attempt has been made to present a hierarchical overlay network in which node joining and failure are handled in an organized fashion according to the capabilities of a node. When a node joins, the

position of the node in the system is determined with its capability. As a result of this a hierarchical structure is formed where high capable nodes are placed at the top level of hierarchy. Beacause of this kind of organized structure, the number of unnecessary messages gets reduced and the query success rate gets increased. With n nodes in the system and TTL of $log_2 n$, the query success rate in HUP is at least 50% (which is only 7% in case of Gnutella). The kind of policy adapted for routing the queries regulates the new nodes which are greedy to consume the resources. Based on the findings, a comparison of HUP with different systems is also presented in Table 5.9.

Chapter 6

Incentive Mechanism in a Hybrid Architecture

In a p2p system the resources are shared among peers without any central authority. According to several Internet Service Providers, more than 50% of Internet traffic is due to p2p applications, sometimes even more than 75% [BYL09]. The continuous growth of Internet in terms of user and bandwidth is accompanied by increasing requirements of a diversified wealth of applications.

Free-riding and white-washing are two severe problems of the p2p networking system. A free-rider is a temporary peer in the network which does not contribute any resources but consumes resources freely from the network. Free-riders create many problems in the network such as it may drop queries made by other. A free-rider contains either zero content or some fake contents. As there may be huge number of free-riders and very less contributors, it may lead to saturation of bandwidth for execution of queries. It may also create unnecessary congestion in the network. As growing number of peers are becoming free-riders, the system may start losing its p2p spirit and become a simple traditional client-server system. Similarly, a white-washer is like a free-rider which frequently leaves the system and re-appears with a different identity to get rid of penalties imposed by the network. It is a challenging problem to turn free-riders and white-washers

into resource contributors. Interestingly many incentive mechanisms are designed to meet such challenges. There is always a need to find out a better mechanism of incentive to handle such problems in p2p systems.

Peer-to-peer networking system is based on the contribution made by the peers present in the network. Normally peers are reluctant to contribute their resources due to number of reasons, such as cost of bandwidth, security threat, slowing down of self downloading process etc. Hence, the objective of incentive mechanism is to encourage peers for contributing resources in p2p system.

In this chapter, an incentive mechanism is described which tries to eliminate free-riders. It introduces better incentive mechanism with an effective grading system of nodes at different hierarchies and exchange of currency called P$ for data exchange. Incentive is a continuous process for smooth running of the system and so user should update resources time to time. The system architecture is discusses in section 6.1. Section 6.2 explains the algorithm for query processing. Experimental results are presented in section 6.3. In section 6.4 related works are discussed and finally, the chapter is summarized in section 6.5.

6.1 System Model

Increased popularity of widely deployed p2p system drifted the attention of p2p research to handle challenges that are caused by free riders, high churn rate and heterogeneity among nodes. In the proposed system, the nodes are classified into three different categories based on their contribution, performance level, and availability in the network. The of nodes are taxonomized as: (i) Normal node (NN), (ii) Stable node (SN) and (iii) Fully-stable node (FS).

i. **Normal node**: Newly joined node added at the lower level of hierarchy is considered as normal node. It cannot contribute much to the p2p network because of its limited bandwidth, processing power and storage capabilities. Stable nodes continuously monitor the behavior of normal nodes. Normal

nodes with good performance and high contribution are promoted to the next level in the hierarchy. A normal node may be connected to more than one stable node based on its contribution level.

ii. **Stable node**: This node is expected to stay in the network with high probability. It is responsible for the registration of normal nodes and also mediates the exchange of currency among normal nodes. The capability of stable node is more than that of normal node. Hence its access level in the network is high. A stable node keeps information about normal nodes and stores all file indexes stored by normal nodes connected to it. Stable node decides the cost of exchange of the data among the normal nodes and maintains the accounts of the normal nodes.

iii. **Fully-stable node**: A fully-stable node has ideal characteristics and available for almost all the time in the network. These nodes are responsible for maintenance of stable nodes, up-gradation of normal nodes to stable nodes, promotion of stable nodes to fully-stable nodes. A fully-stable node is responsible for generating the currency i.e. $P\$$ and is used in inter-cluster communication.

Figure 6.1 shows the proposed 3-tier hybrid overlay structure in which first, second and third level of hierarchy contains fully-stable nodes, stable nodes and normal nodes, respectively. The network is divided into several clusters. Nodes in a cluster are topologically close to each other. Hence, intra-cluster communication cost is less. Each cluster is formed with stable nodes where normal nodes are connected. Stable nodes in a cluster are connected with unstructured network. Since number of stable nodes is very less compared to the number of normal nodes, any flooding based technique causes less overhead to retrieve data and overcome the restriction imposed on unstructured network due to time-to-live (TTL) value. A stable node may be connected with more than one fully-stable nodes to avoid single point of failure. A normal node is directly connected to one

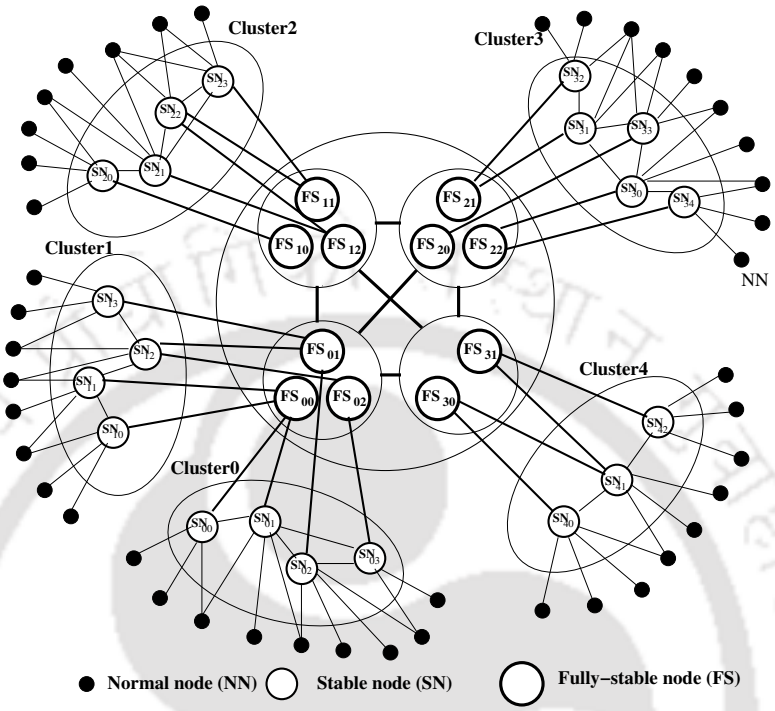

Figure 6.1: A high level architecture of hybrid p2p network

or more stable nodes and hence does not require any overhead in case of high churn rate.

Figure 6.2 shows the first level of proposed hybrid overlay structure where fully-stable nodes are divided into different clusters. Fully-stable nodes of a cluster may be connected with several stable nodes of different clusters. The clusters of fully-stable node are connected directly using mesh-like topology. Fully-stable nodes within a cluster are also directly connected to each other. Fully-stable nodes are used for inter-cluster communication in the first level of hierarchy. For example, $\forall\ i$ and $\forall\ j\ FS_{ij} \in G_i$, FS_{ij}s are connected with mesh topology within cluster G_i. Fully-stable nodes FS_{02}, FS_{12}, FS_{20} and FS_{30} are also connected with each other for inter-cluster communication. Since the network uses a specific structure at higher layer and flooding is used at lower layer, the system is termed as hybrid.

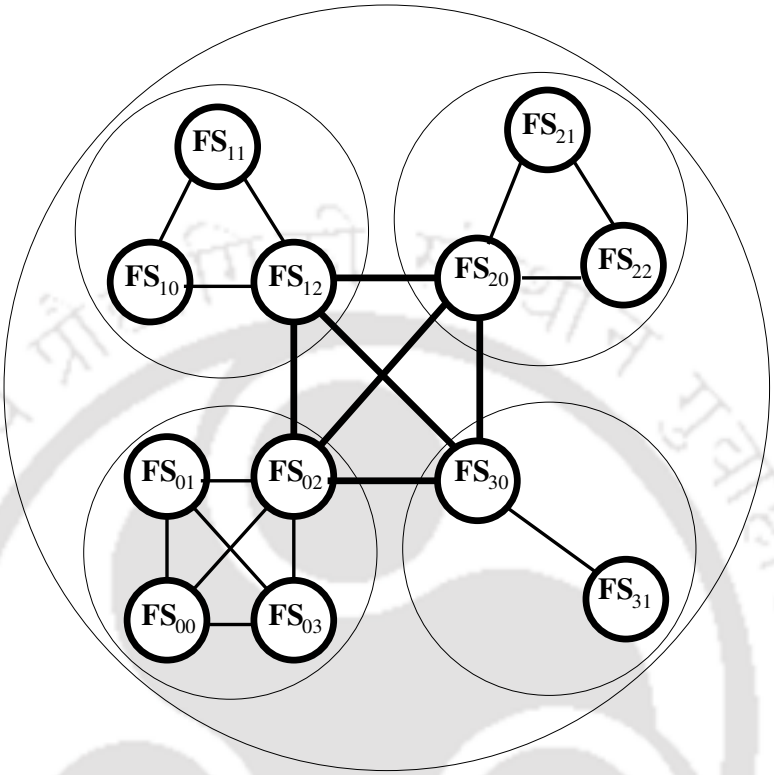

Figure 6.2: First level of Figure 6.1

The proposed hybrid overlay structure has several advantages over existing ones.

i. Since newly joined nodes are connected at the third level of hierarchy, it does not cause any overhead in case of frequent joining and leaving of nodes.

ii. It provides responsibilities to nodes according to their capabilities and availability in the network.

iii. Since, nodes within a cluster are topologically close to each other, overhead to communicate among nodes in a cluster is reduced.

iv. It motivates nodes to share more resources to survive in the network.

6.1.1 Bootstrapping

It is assumed that some pre-occupied fully-stable nodes are available in the system. The first m nodes that join the network become stable nodes and the rest (that join later) are considered as normal nodes. However, number of fully-stable nodes and stable nodes may increase further to balance the load or to increase the access level of existing nodes. When a node in a new cluster wants to join the network, it can be connected with one of the existing clusters of fully-stable nodes and further they can create their own cluster of fully-stable nodes.

6.1.2 Registration Service

When a normal node wants to join one of the cluster in the network it has to register with at least one of the stable nodes present in the cluster. However, a normal node may connect with more than one stable node by increasing its contribution to the network.

Once the normal node gets registered with one or more stable nodes it has to publish all its shared file indexes to stable nodes. The stable nodes assign certain amount of $P\$$ based on the contribution of the normal node. The $P\$$ is used for every operation performed by the node. A node earns certain amount of $P\$$ for every service it provides to the network.

Stable nodes continuously observe the behavior and performance of normal nodes to categorize them in one of the grades i.e. grade-0, grade-1 and grade-2.

i. Grade-0: Recently joined nodes are considered in grade-0. These nodes may have rational behavior and may be greedy to consume resources. These nodes have to share minimum resources and have to pay nominal registration charge. These nodes are regularly monitored by stable nodes. If their behavior is found good, they can be promoted to grade-1.

ii. Grade-1: Nodes with average performance and contribution level are kept

in grade-1. Their access to resources in the system is more in comparison to grade-0 nodes.

iii. Grade-2: Grade-1 node is promoted to grade-2 node with higher availability and performance. Grade-2 nodes are the best candidate for stable nodes in future. Each stable node maintains a list of grade-2 nodes, so that it can recommend some of them to fully-stable nodes as candidates for stable node. Similarly, each fully-stable node maintains a list of grade-2 stable nodes which can be promoted to fully-stable nodes whenever necessary.

Within a cluster, stable nodes may be connected with several normal nodes. Stable nodes recommend some of the grade-2 normal nodes as candidates for new stable node. A fully-stable node upgrades these recommended normal nodes to stable nodes based on certain criteria. Stable nodes publish their file indexes to fully-stable nodes of that cluster. These indexes are used by the fully-stable node for either retrieving files for that cluster or for testing the stable node. The fully-stable nodes categorize stable nodes into grade-0, grade-1 and grade-2. If a fully-stable node connected to a cluster gets overloaded, then it can promote a stable node of grade-2 to fully-stable node.

6.1.3 Avoiding Free Riders

It is mentioned in section 6.1.2 that if a node wants to join the network it has to share minimum resources and nominal registration charge. Minimum resource requirement is low enough to encourage a node to easily join the network and registration service charge is enough to avoid whitewashing problem.

Grading of normal nodes is done by stable nodes as grade-0, grade-1 and grade 2 based on their contribution level, performance and availability in the network. Contribution level and performance are measured in terms of amount of resources shared and percentage of requests satisfied by a normal node, respectively.

In this approach, for every unit of data query a node has to spend some amount of $P\$$ which are given to the node at the time of registration by the stable node. As node stays in the network and provides service to the other nodes it can earn $P\$$. Only those nodes with enough $P\$$ can access data in the network. Nodes which provides different kinds of data which are in need of many users can earn more. This scheme encourages nodes to contribute more and perform well to get high access to the resources.

6.1.4 Stabilization

A node is free to leave at any time from the network. There are two types of leaving, intentional and failure (due to the system crash or power failure). Normal node can simply leave the network after informing stable nodes where it is connected. If a stable node wants to leave the system, then the place of that stable node is taken by one of its normal node of grade-2. It sends all its indexes to newly joined stable node. When a fully-stable node wants to leave the network it upgrades one of the stable nodes of grade-2 (that is connected with it) to fully-stable node.

For failure detection and handling, the fully-stable nodes send *keep_alive* messages to other fully-stable nodes and stable nodes connected to it. A fully-stable node assigns responsibilities of the failed nodes to other node(s) if it does not get reply within a specified time. Stable nodes also send *keep_alive* messages to their normal nodes. If any normal node leaves, then it deletes all indexes related to it.

6.2 Query Processing

The system sends communication messages among nodes for query execution. Query processing involves query request message from normal node to stable node, stable node to stable node, stable node to fully-stable node, fully-stable node to stable node and fully-stable node to fully-stable node.

When a normal node NN_1 requests for a file with index X to the stable node SN_{ij}, then it may get node list as NN_2, NN_3 and NN_4. NN_1 can request to any one of these nodes to download the file. Let NN_1 download the file from NN_2 then NN_1 reports to SN_{ij} for downloading file having index X from a node NN_2. Then NN_2 reports to its stable node about delivery of the file index X to NN_1. Now account of NN_2 being credited as per the charged policy prevailed. Data transfer takes place if the normal node has enough balance to download. Query request of a stable node and a fully-stable node is handled in similar manner.

The description of related algorithms are stated below.

- **Algorithm-**1: This is executed when a normal node, NN_l, requests a file with index, X, to the stable node, SN_{ij}. It returns a list of node *id*s with index X.
  - Step-1: The stable node SN_{ij} processes this request by checking the account of the normal node, NN_l, for minimum number of P$ required for searching.
  - Step-2: If NN_l has enough P$ then search progresses. Otherwise it returns a $NULL$ list.
  - Step-3: If the index is found in the connected normal nodes, stable node returns a list of node *id*s to NN_l which contains the index X.
  - Step-4: The stable node SN_{ij} floods the query request to all the neighbor stable nodes in its cluster. The neighbor stable nodes process algorithm 2 and returns corresponding node list to the normal node NN_l.
  - Step-5: If index X is not found then it returns a $NULL$ list.
- **Algorithm-**2: This is executed by a stable node SN_{ij} with index X. It returns a list of node *id*s with index X.

- Step-1: The stable node sends query request with index X to all the neighbor stable nodes in its cluster. All stable nodes check index X in their file index list. If found then it returns a list of node *id*s to SN_{ij}.
- Step-2: The stable node forwards the request to the fully-stable node FS_{ij} to which it is connected. The fully-stable node executes algorithm 3 and returns corresponding node list to the stable node SN_{ij}.
- Step-3: If index X is not found, then it returns a $NULL$ list.

- **Algorithm-**3: This is executed by a fully-stable node FS_{ij} with index X. It returns a list of node *id*s with index X.
  - Step-1: The fully-stable node FS_{ij} sends the query request to stable nodes which are not belong to cluster of stable node SN_{ij}. These stable nodes execute algorithm 2 and returns corresponding node list to the fully-stable node FS_{ij}.
  - Step-2: The fully-stable node sends query request with index X to all fully-stable nodes in the system. All fully-stable nodes check index X in their file index list. If found, then it returns a list of node *id*s to FS_{ij}.
  - Step-3: If index X is not found, then it returns a $NULL$ list.

6.2.1 Malicious Behavior of Nodes

It is assumed that fully-stable nodes have ideal characteristics and hence can not be malicious. Whereas behavior of stable nodes and normal nodes can be under consideration. In this approach, stable nodes or fully-stable nodes provide a list of nodes with resources requested by *querier node*. When the querier node reports that it does not consume any resources even though it has done so, a new list of grade-2 nodes with that resource are again provided. If the querier node still reports that grade-2 nodes does not collaborate it in getting resources, then the

querier node is marked as malicious and its requests are not encouraged for a specific period of time.

Few malicious nodes may intentionally support each other to increase their performance level. For example, NN_1 may ask only those files that are with NN_2 and would report received resource from NN_2 to upper level node and vice versa. In this way these nodes try to increase their $P\$$ value. This problem can be solved by providing different list of nodes each time a node makes a query. For example, if NN_2, NN_3, NN_4, NN_5, NN_6 are the list of nodes which hold a particular resource X. For the first request stable nodes or fully-stable nodes may return node list NN_2, NN_3 and NN_4; while, for second request of same file it may return NN_4, NN_5 and NN_6.

6.2.2 Load Balancing

It provides a very simple mechanism to handle the load in network. It keeps minimum contribution level for each type of nodes. Since, a node can not share data more than its capacity there is no chance of overloading of that node due to shared data. However, a stable node or fully-stable node may be overloaded due to the request received from other nodes and hence they may not satisfy all requests. In this case, some of the normal nodes and stable nodes of grade-2 are promoted to become stable nodes and fully-stable nodes, respectively to share the load.

6.3 Experiment Results

The algorithms discussed here are implemented considering the above mentioned hybrid structure. The p2p structure is constructed during the bootstrapping process. It can perform joining of a node, update of a node and execution of queries. During joining of a node, a new node joins as a normal node. The new node is assigned some grade value in terms of $P\$$. The new node must upload

some resources to the system. The grade values of the new node are updated according to the number of resources added. During upload operation an existing node is allowed to add more resources to the system and accordingly grading of node is adjusted. An existing node executes a query with required resource *id*. The querier node receives a list of node *id*s containing desired resource *id*. The querier node selects one node from the list to get the resource. The grade value of the querier node is decreased and the grade value of the respondent node is increased.

No of events	No of Grade-0	No of Grade-1	No of Grade-2
0	672	0	0
1000	450	200	351
2000	353	294	676
3000	308	343	980
4000	308	377	1289
5000	340	423	1559
6000	374	480	1801
7000	414	536	2047
8000	445	595	2291
9000	491	653	2498
10000	540	713	2734

Table 6.1: Number of nodes having Grade-0, Grade-1 and Grade-2 during each multiples of 1000 iterations

6.3.1 Event-Grade Analysis

Here, one event is performed in each iteration. Outputs are obtained at multiple of 1000 events. The observations are represented in Table 6.1 which shows the number of nodes having grade-0, grade-1 and grade-2 during each iteration.

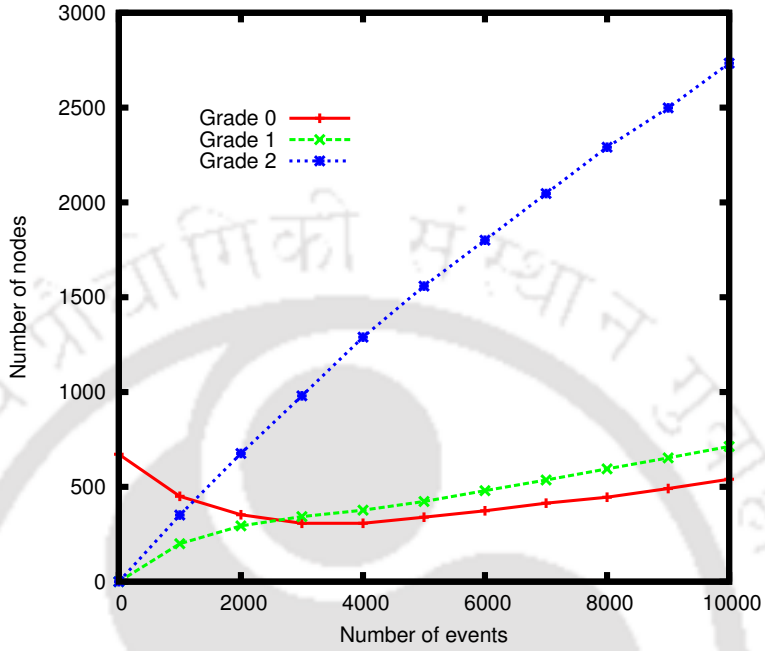

Figure 6.3: Event Grade Analysis

Figure 6.3 shows the number of grade-0, grade-1 and grade-2 nodes. There are n number of grade-0 nodes during bootstrapping. This number is chosen empirically as 672 nodes where 512 are normal nodes, 128 are stable nodes and 32 are fully-stable nodes. Hence during instantiation of the network there are no nodes having grade-1 and grade-2. Ideally, numbers of grade-0 nodes should be decreased as some of them are turning into grade-1 and grade-2 by providing resources in the network. But it is not decreased gradually because of joining of new nodes in the system. Hence it is seen that grade-0 graph is decreased up to a point ($P\$$) during event numbers 3000 to 4000, after that it starts increasing. It is observed from the graph that grade-1 and grade-2 nodes increase gradually by providing resources and responding queries made by others. A node with a higher grade value has high accessing power in the network and so every node tries to place itself in higher grade by providing more resources.

No of events	No of success queries	No of failed queries
1000	319	3
2000	658	9
3000	992	14
4000	1322	16
5000	1652	18
6000	1972	20
7000	2289	20
8000	2619	21
9000	2947	22
10000	3295	23

Table 6.2: Numbers of nodes for which queries are successful or failed during each multiples of 1000 iterations

6.3.2 Event-Query Analysis

Output of query processing is shown in Table 6.2. The graph in Figure 6.4 shows the total queries performed and number of successful and failed queries in the system. The query may be failed due to unavailability of the resource in the network or due to lower grade value of the querier node than the resource holder. It is observed from the graph that because of higher participation of nodes only a few queries are failed. This indicates proper functioning of incentive mechanism in the system.

6.3.3 Malicious Behavior of Nodes

It is observed from the Table 6.3 that a node executing a query gets a list of node *id*s containing the desired data. The querier node gets a list selected randomly

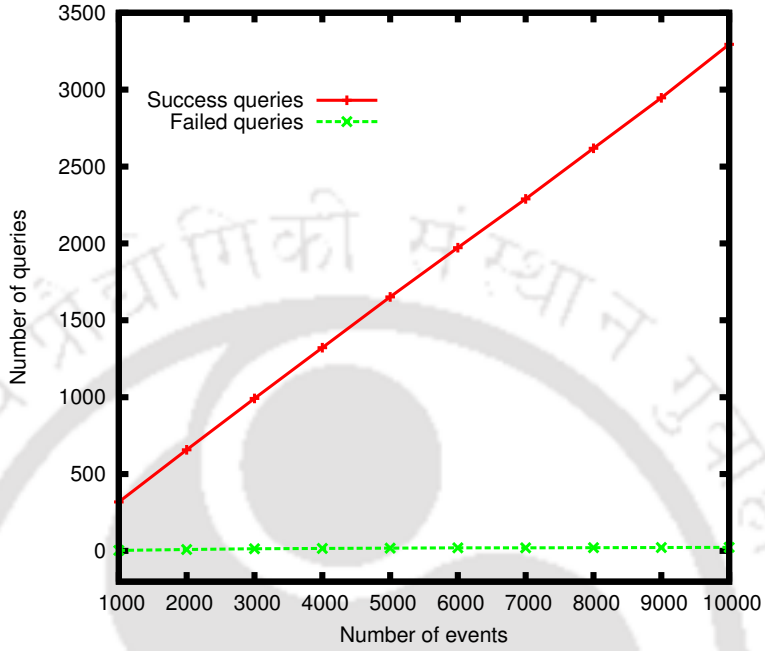

Figure 6.4: Event Query Analysis

from that list. Every time whenever a node makes a query it may get a different list of node *ids*.

6.3.4 Grade Value Analysis

It is observed from the Table 6.4 that initially assigned $p\$ = 100$ is either increased or decreased. Figure 6.5 represents graphs for $P\$$ as it changes after 1000 iterations. The value of $P\$$ is increased for providing resources to other node and it is decreased for accessing resources from the system.

The grading of node changed according to the nature of $P\$$ earned or spent. It is observed that some of the nodes in the system can upgrade to higher grade according to the need of the system.

Queried Node *id*	Search Key	Search Key found in Node *id*s
5	960	98, 350, 5, 104, 206, 11, 25, 277, 13, 27, 407, 13, 60, 440, 31
54	765	95, 192, 2, 105, 357, 12, 106, 358, 13, 35, 70, 18
145	478	25, 5, 710, 3, 101, 354, 10, 278, 13
273	980	63, 662, 4, 98, 357, 7, 74, 147, 15
970	12	7, 115, 495, 22

Table 6.3: List of node *id*s found for one Search key value

6.3.5 Load Balancing

Load balancing in the system is performed whenever a node gets overloaded. In that case, a grade-2 node can be used to share resources of the overloaded node. The simulation result shows two lists where a normal node can be upgraded to a stable node and a stable node can be upgraded to a fully-stable node. The list of normal nodes which are qualified for up-gradation to stable nodes is shown in Table 6.5.

6.4 Related Works

Anceaume et al. [AGR05] propose a middleware architecture to optimize the resources in p2p network. Middleware layer includes four different services: co-operation tracking, aggregation service, semantic group membership and registration service. This scheme provides fair resource sharing mechanism in a large

Node *id*	Initial *P*$	*P*$ after 1000 steps	*P*$ after 2000 steps	*P*$ after 3000 steps	*P*$ after 4000 steps	*P*$ after 5000 steps	*P*$ after 6000 steps	*P*$ after 7000 steps	*P*$ after 8000 steps	*P*$ after 9000 steps	*P*$ after 10000 steps	Grade
0	100	300	275	325	325	325	325	325	325	325	325	0
1	100	1350	1350	1350	1650	1925	3025	4275	4275	4275	4250	2
2	100	875	875	875	850	1100	1100	1500	1500	2500	2500	2
3	100	150	325	625	625	1425	1425	1425	1700	1700	1700	2
4	100	200	325	325	325	675	675	675	675	750	725	1
5	100	500	1100	2100	2100	2075	2850	3325	3325	4050	4050	2
6	100	100	75	75	375	350	325	325	1375	1375	1350	2

Table 6.4: Changing of grading with reference to *P*$ spent

Node *id*	Amount of *P*$
44	1250
120	1525
153	1275
318	1300
673	2125
675	1400

Table 6.5: List of NN qualified for up-gradation to SN

scale dynamic p2p network with rational users. Ham et al. [HA05] introduced a credit system to encourage nodes to contribute more in p2p network. They also discuss the mechanism to keep the credit values consistent in the presence of malicious nodes and without requiring any centralized server. They assume the existence of p2p overlay structure to locate resources and handle queries for those resources. Feldman et al. [FPCS04] present an economic model to avoid free riding and whitewashing problem. They assume that the cost of contribution is the inverse of the total percentage of contribution and finds that when penalty is applied on newly joined nodes to handle whitewashing problem then perfor-

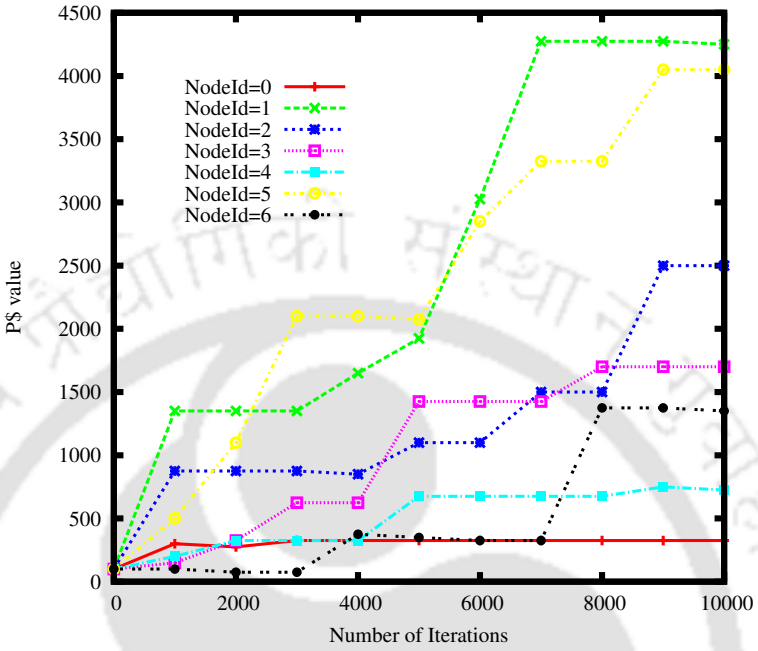

Figure 6.5: Graph shows P\$ changing after evry 1000 steps

mance degrades only when the turnover rate among the nodes is high. Feldman et al. [FC05] discuss some open questions related to free-riding in p2p systems: factors affecting the degree of free-riding, incentive mechanisms to encourage user co-operation and challenges in the design of incentive mechanisms for p2p systems. Yung et al. [YZLD05] propose a two step approach to minimize free riding; mechanisms to distribute queries to all the sources and encourage nodes to make their resources available. In this approach, a selfish behavior is demonstrated more by short online time. Buragohain et al. [BAS03] use game theory to study the behavior of peers and propose an incentive scheme based on differential services to eliminate free riding in p2p system. Karakaya et al. [KKU04] propose a measurement based approach to reduce free riding in unstructured p2p system; their scheme requires each peer to monitor their neighbor peers to control free riding. In our system, we have proposed a 3-tier hierarchy to deal with the issues like efficient searching, network management, eliminating free riding and

whitewashing. Pianese et al. [PPKB07] present an unstructured mesh-based p2p system to support live streaming to large audiences under the arbitrary resource availability as is typically the case for Internet. It constantly optimizes strategy that is based on pairwise incentives.

6.5 Summary

An attempt has been made to present a 3-tier hybrid overlay structure that can cope up with high churn rate of newly joined nodes. The proposed approach differentiates the role of powerful nodes to others by classifying the nodes into three categories: fully-stable node, stable node and normal node. As number of stable nodes are less than the number of normal nodes any flooding based technique causes less overhead to retrieve data and also overcome with the restriction imposed on unstructured network due to time-to-live (TTL) value. Nodes within a cluster are topologically close to each other, hence intra-cluster communication cost will be less. It is evident from the experimental results that this incentive scheme encourages nodes to contribute more and perform well in a p2p network.

Chapter 7

Conclusion and Future Works

In this work, we have made an attempt to improve efficiency of p2p systems by proposing various mechanisms considering different aspects of structured and unstructured systems. Existing p2p systems are entirely flat in which all the participating peers assume equal role in the network. For resource allocation and discovery, these systems use either a centralized server or a distributed hash table that is evenly distributed among all the participating peers. However, such systems do not scale well with increase in the number of peers and cannot not cope up with frequent joining and leaving of peers in the real world. Introduction of hierarchy into a p2p system is found to be inevitable to make the system more scalable and efficient. It has been observed that if the responsibility of maintaining the overlay network lies with the more stable peers, then the network as a whole will become stable. In this context, we have been inspired to classify nodes on the basis of different parameters and place them at different levels of hierarchies. We now summarize the major contributions of our work and then give some possible future extensions to our work.

7.1 Major Contributions

The major contributions of this dissertation involve in proposing new hierarchical overlay systems, one each for structured and unstructured; and development of two schemes: one for handling churn rate in Chord and the other for removing free-riders by adopting an appropriate incentive mechanism. Towards the development of these frameworks, the important contributions of our work are outlined below.

- For the development of the hierarchical overlays for both structured and unstructured systems we have proposed classification of nodes into different categories based on different parameters like duration of stay in the network, capability, etc. This helps placing the nodes at different levels of hierarchy based on their category to exploit full advantage resulting in improved performance and maintainability.

- We have proposed a structured hierarchical peer-to-peer (SHP) in Chapter 3, which can effectively handle high churn rate. Based upon the heterogeneity, nodes were divided into three categories. The extra effort to store the routing tables of short living nodes in the system are avoided, thus reducing a substantial amount of messages. It has been shown (ref. section 3.3.1) that if the rate of joining and leaving of nodes are high, SHP saves $2 \times \lambda_j \times (\log n)^2 \times T_{avg}$ number of updates compared to Chord. In average case, SHP saves about 66.67% messages in repairing the routing table compared to Chord. It also provides efficient solution for handling both point query and range query. To the best of our knowledge no DHT-based structured overlay system provides efficient and scalable solution to range query. We have also proposed a method to tackle the problem of load balancing in this system. It has been shown that if $\frac{n}{g}$ is the average number of nodes per group, then the cost of load balancing is $O(\log \frac{n}{g})^2$.

- A modification to the Chord protocol is presented to handle churn rate

in the system in an efficient manner. We introduce temporal concept to classify nodes into three different categories. This system is shown to be more suitable under the occurrence of frequent joining and leaving of nodes.

- We present an organized network architecture for unstructured peer-to-peer systems where nodes are added to the network in a systematic way to efficiently utilize the resources of the nodes. It has been shown that the number of unnecessary messages gets reduced and the success rate increases substantially. With n nodes in the system and TTL of $log_2 n$, the query success rate in HUP is at least 50%. The kind of policy adopted for routing the queries regulates the new nodes which are greedy to consume the resources.

- An incentive mechanism has been proposed to eliminate the problem of free-riders. It introduces better incentive mechanism with an effective grading system of nodes at different hierarchies.

- All these methods have been simulated using suitable tools developed in the context and experimental results have been used to substantiate our claims made in each case.

7.2 Future Works

Many p2p networks have been deployed on the Internet these days, and some of them have become very popular. But some open problems (e.g. scalability, performance, load-balancing, security and QoS) are still on the way for the success of many p2p overlay networks. We briefly outline here some of the possible future extensions to our work.

- The hierarchical systems that we have developed can be used to build a working models for the benefit of p2p users. We believe that with the diverse advantages offered these can be highly useful component and platform for large scale distributed applications.

- The modification of Chord suggested in chapter 4 can be integrated with existing Chord overlay to build a new overlay system. The applications that suffers from performance loss due to high churn rate will be able to cope up with high system load.

- The incentive mechanism that has been proposed can be a basis for implementation of a practical economic model.

- In this dissertation, we have restricted our study to improve performance of p2p systems. All the proposed systems can be extended further to handle various security problems and to include mechanisms to provide appropriate QoS based on the demand of a particular application.

- The ideas proposed here will continue to enhance further. The large test bed can be created by selecting servers of Internet Service Provider as fully-stable nodes and specific regions as clusters. In most instances an end-user is interested in looking information within their region (i.e. most of the cases teaching materials, songs, videos etc. are retrieved from local servers of a region).

Bibliography

[Abe01] K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In *Proceedings of the 9th International Conference on Cooperative Information Systems*, pages 179–194, Trento, Italy, 2001. Springer-Verlag.

[ACK+02] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Communication, ACM*, 45(11):56–61, 2002.

[ACMD+03] K. Aberer, P. Cudr-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-grid: A self-organizing structured p2p system. *ACM SIGMOD Record*, 32(3), 2003.

[AGR05] E. Anceaume, M. Gradinariu, and A. Ravoaja. Incentives for p2p fair resource sharing. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 253–260, Washington, DC, USA, 2005. IEEE Computer Society.

[AH00] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5, 2000.

[BAH+06] R. Blanco, N. Ahmed, D. Hadaller, L. G. Sung, H. Li, and M. A. Soliman. A survey of data management in peer-to-peer systems.

Technical report, David R. Cheriton School of Computer Science, University of Waterloo, Canada, Jun 2006.

[BAS03] C. Buragohain, D. Agrawal, and S. Suri. A game theoretic framework for incentives in p2p systems. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, pages 48–56, Washington, DC, USA, 2003. IEEE Computer Society.

[BAS04] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. *SIGCOMM Computer Communication Review*, 34(4):353–366, 2004.

[BCRS08] S. Burresi, C. Canali, M. E. Renda, and P. Santi. Meshchord: A location-aware, cross-layer specialization of chord for wireless mesh networks (concise contribution). In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications*, pages 206–212, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[BYL09] J. F. Buford, H. Yu, and E. K. Lua. *P2P Networking and Applications*. Morgan Kaufmann, 2009.

[CCF04] M. Cai, A. Chervenak, and M. Frank. A peer-to-peer replica location service based on a distributed hash table. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 56, Washington, DC, USA, 2004. IEEE Computer Society.

[CCR04] M. Castro, M. Costa, and A. Rowstron. Peer-to-peer overlays: structured, unstructured, or both? Technical report, Microsoft Research, USA, 2004.

[CKGM04] T. Condie, S. D. Kamvar, and H. Garcia-Molina. Adaptive peer-to-peer topologies. In *Proceedings of the Fourth International Con-*

ference on Peer-to-Peer Computing, pages 53–62, Washington, DC, USA, 2004. IEEE Computer Society.

[CLGS04] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundaram. Querying peer-to-peer networks using p-trees. In *Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS 2004*, WebDB '04, pages 25–30, New York, NY, USA, 2004. ACM.

[CMM02] R. Cox, A. Muthitacharoen, and R. Morris. Serving dns using a peer-to-peer lookup service. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 155–165, London, UK, 2002. Springer-Verlag.

[Coh03] B. Cohen. Incentives build robustness in bittorrent. Technical report, bittorrent.org, 2003.

[CSWH00] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Lecture Notes in Computer Science*, pages 46–66, 2000.

[DKK+01] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 202–215, New York, NY, USA, 2001. ACM.

[FC05] M. Feldman and J. Chuang. Overcoming free-riding behavior in peer-to-peer systems. *ACM Sigecom Exchanges*, 6:41–50, 2005.

[FPCS04] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica. Free-riding and whitewashing in peer-to-peer systems. In *Proceedings of the third Annual Workshop on Economics and Information Security*, pages 228–136, may 2004.

[GAA03] A. Gupta, D. Agrawal, and A. E. Abbadi. Approximate range selection queries in peer-to-peer systems. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR)*, Jan 2003.

[GBL+03] I. Gupta, K. Birman, P. Linga, A. Demers, and R. V. Renesse. Kelips: building an efficient and stable p2p dht through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS 03)*, 2003.

[GEBF+03] L. Garces-Erce, E. Biersack, P. Felber, K. Ross, and G. Urvoy-Keller. Hierarchical peer-to-peer systems. In *Proceedings of the ACM/IFIP International Conference on Parallel and Distributed Computing (Euro-Par)*, pages 643–657, 2003.

[GS04] J. Gao and P. Steenkiste. An adaptive protocol for efficient support of range queries in dht-based systems. In *Proceedings of the 12th IEEE International Conference on Network Protocols*, pages 239–250, Germany, 2004.

[HA05] M. Ham and G. Agha. Ara: A robust audit to prevent free-riding in p2p networks. In *Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing*, pages 125–132, 2005.

[HJS+03] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: a scalable overlay network with practical locality properties. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, page 9, Berkeley, CA, USA, 2003.

[HKF+06] S. B. Handurukande, A. M. Kermarrec, F. L. Fessant, L. Massoulié, and S. Patarin. Peer sharing behaviour in the edonkey network, and implications for the design of server-less file sharing systems. In

Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006, pages 359–371, New York, NY, USA, 2006. ACM.

[HKO05] B. Hudzia, M. T. Kechadi, and A. Ottewill. Treep: A tree based p2p network architecture. pages 1–15, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[HLYW05] F. Hong, M. Li, J. Yu, and Y. Wang. Pchord: Improvement on chord to achieve better routing efficiency by exploiting proximity. In *Proceedings of the First International Workshop on Mobility in Peer-to-Peer Systems*, pages 806–811, Washington, DC, USA, 2005. IEEE Computer Society.

[JW07] Yuh-Jzer Joung and Jiaw-Chang Wang. Chord2: A two-layer chord for reducing maintenance overhead via heterogeneity. *Computer Networks*, 51(3):712–731, 2007.

[JY06] Y. Jiang and J. You. A low latency chord routing algorithm for dht. In *Proceedings of the 1st International Symposium on Pervasive Computing and Applications*, volume 2, pages 825–830, 2006.

[KBC+00] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, 2000.

[KBH05] G. Kunzmann, A. Binzenhofer, and R. Henjes. Analyzing and modifying chord's stabilization algorithm to handle high churn rates. In *Proceedings of the IEEE International Conference on Networks*, volume 2, page 6, 2005.

[KK03] M. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Peer-to-Peer systems workshop*, pages 98–107, Berkeley, CA, USA, 2003.

[KKU04] M. Karakaya, I. Krpeoglu, and Z. Ulusoy. A distributed and measurement-based framework against free riding in peer-to-peer networks. In *Proceedings of the IEEE International Conference on Peer-to-Peer Computing*, pages 276–277, 2004.

[LCP+05] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys Tutorials, IEEE*, 7(2):72–93, Feb 2005.

[LHL09] E. J. Lu, Y. Huang, and S. Lu. Ml-chord: A multi-layered p2p resource sharing model. *J. Network and Computer Applications*, 32(3):578–588, 2009.

[LHLH03] Z. Li, D. Huang, Z. Liu, and J. Huang. Research of peer-to-peer network architecture. In *Proceedings of the International Conference on Communication Technology*, volume 1, pages 312–315, 2003.

[LKR06] J. Liang, R. Kumar, and K. Ross. The fasttrack overlay: a measurement study. *Computer Networks*, 50:842–858, 2006.

[LLZ07] H. Liu, P. Luo, and Z. Zeng. A structured hierarchical p2p model based on a rigorous binary tree code algorithm. *Future Generation Computer System*, 23(2):201–208, 2007.

[LMR02] N. A. Lynch, D. Malkhi, and D. Ratajczak. Atomic data access in distributed hash tables. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 295–305, London, UK, 2002. Springer-Verlag.

[LRW03] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the kazaa network. In *Proceedings of the Third IEEE Workshop on Internet Applications*, pages 112–120, 2003.

[LTZ08] Y. M. Li, T. Tan, and Y. P. Zhou. Analysis of scale effects in peer-to-peer networks. *IEEE/ACM Transaction on Networking*, 16(3):590–602, 2008.

[LX06] Z. Li and G. Xie. A distributed load balancing algorithm for structured p2p systems. In *Proceedings of the 11th IEEE Symposium on Comnputers and Communuications*, pages 417–422, Washington, DC, USA, 2006. IEEE Computer Society.

[LY08] J. Li and X. Yang. An optimized chord algorithm for accelerating the query of hot resources. In *Proceedings of the International Symposium on Computer Science and Computational Technology*, pages 644–647, Washington, DC, USA, 2008. IEEE Computer Society.

[LYHL06] J. W. Lin, M. F. Yang, C. Y. Huang, and C. T. Lin. Reliable hierarchical peer-to-peer file sharing systems. In *Proceedings of the IEEE conference (TENCON 2006)*, pages 1–4, 2006.

[LZ06] J. Liu and H. Zhuge. A semantic-based p2p resource organization model r-chord. *Journal of Systems and Software*, 79(11):1619–1634, 2006.

[MBR03] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: distributed hashing in a small world. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, pages 10–10, Berkeley, CA, USA, 2003. USENIX Association.

[MM02a] N. Maibaum and T. Mundt. Jxta: A technology facilitating mobile peer-to-peer networks. In *Proceedings of the International Mobility and Wireless Access Workshop*, pages 7+, 2002.

[MM02b] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.

[MMO06] M. Marzolla, M. Mordacchini, and S. Orlando. Tree vector indexes: efficient range queries for dynamic content on peer-to-peer networks. In *Proceedings of the Parallel, Distributed, and Network-Based Processing*, 2006.

[MNR02] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192, New York, NY, USA, 2002. ACM.

[NBLR06] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai. A survey of peer-to-peer network simulators. In *Proceedings of The Seventh Annual Postgraduate Symposium*, 2006.

[NT04] N. Ntarmos and P. Triantafillou. Seal: Managing accesses and data in peer-to-peer sharing networks. In *Proceedings of the Peer-to-Peer Computing*, pages 116–123, 2004.

[PPKB07] F. Pianese, D. Perino, J. Keller, and E. W. Biersack. Pulse: An adaptive, incentive-based, unstructured p2p live streaming system. *IEEE Transactions on Multimedia In Multimedia*, 9:1645–1660, Dec 2007.

[PRR97] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, pages 311–320, New York, NY, USA, 1997. ACM.

[Pug90] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Communication, ACM*, 33(6):668–676, 1990.

[pv07a] Gnutella protocol v0.4. http://dss.clip2.com/ gnutellaprotocol04.pdf, 2007.

[pv07b] Gnutella protocol v0.6. http://rfc-gnutella.sourceforge.net/src/rfc-0 6-draft.html, 2007.

[RD01a] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, Nov 2001.

[RD01b] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, Lake Louise, Canada, 2001.

[Ref10a] Webpage Reference. http://mathworld.wolfram.com/ butterflygraph.html, 2010.

[Ref10b] Webpage Reference. http://www.fact-index.com/ g/gr/grokster.html, 2010.

[Ref10c] Webpage Reference. http://www.imesh.com, 2010.

[Rep08] Sandvine Report. 2008 analysis of traffic demographics in north-american broadband networks. Technical report, sandvine.com, 2008.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content- addressable network. In *Proceedings of the ACM SIGCOMM*, pages 161–172, 2001.

[RHB08] R. Ranjan, A. Harwood, and R. Buyya. Peer-to-peer-based resource discovery in global grids: a tutorial. *Communications Surveys Tutorials, IEEE*, 10:6–33, Feb 2008.

[RHKS02] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of the INFOCOM 2002*, volume 3, pages 1190–1199, 2002.

[Rip01] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proceedings of the IEEE 1st International Conference on Peer-to-Peer Computing*, pages 99–100, Linkoping, Sweden, Aug 2001.

[RLS+03] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured p2p systems. In *Lecture Notes in Computer Science*, volume 2735, pages 68–79. Springer Berlin / Heidelberg, 2003.

[Sch01] R. Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings of the First International Conference on Peer-to-Peer Computing*, pages 101–102, Aug 2001.

[SE05] R. Steinmetz and K. Wehrle (Eds.). *P2P Systems and Applications*. Springer-Verlag Berlin Heidelberg, 2005.

[SGG02] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of the Multimedia Computing and Networking*, pages 156–170, 2002.

[SGL04] M. Srivatsa, B. Gedik, and L. Liu. Scaling unstructured peer-to-peer networks with multi-tier capacity-aware overlay topologies. In *Proceedings of the Tenth International Conference Parallel and Dis-*

tributed Systems, page 17, Washington, DC, USA, 2004. IEEE Computer Society.

[SKG07] B. K. Shrivastava, G. Khataniar, and D. Goswami. Range query over structured overlay system. In *Proceedings of the International Conference on Grid Computing & Applications*, pages 137–143, 2007.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM.

[SMLN+03] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *IEEE Transactions on Networking*, 11(1):17–32, Feb 2003.

[SR05] D. Stutzbach and R. Rejaie. Capturing accurate snapshots of the gnutella network. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2825–2830, 2005.

[SSJKQG07] Y. Shao-Shan, Y. Jiong, K. Kamil, and S. Qi-Gang. Dr-chord - an efficient double-ring chord protocol. In *Proceedings of the Sixth International Conference on Grid and Cooperative Computing*, pages 197–202, Washington, DC, USA, 2007. IEEE Computer Society.

[TKLB07] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann. Bubblestorm: Resilient, probabilistic and exhaustive. In *Proceedings of the P2P Search ACM SIGCOMM*, pages 49–60. ACM, 2007.

[TS04] S. A. Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys (CSUR)*, 36(4):335–371, Dec 2004.

[WXZ05] C. Wang, L. Xiao, and P. Zheng. Differentiated search in hierarchical peer-to-peer networks. In *Proceedings of the International Conference on Parallel Processing*, pages 269–276, 2005.

[WYG06] J. Wang, S. Yang, and L. Guo. A bidirectional query chord system based on latency-sensitivity. In *Proceedings of the Fifth International Conference on Grid and Cooperative Computing*, pages 164–167, Washington, DC, USA, 2006. IEEE Computer Society.

[XCK03] D. Xuan, S. Chellappan, and M. Krishnamoorthy. Rchord: an enhanced chord system resilient to routing attacks. In *Proceedings of the International Conference in Computer Networks and Mobile Computing*, pages 253–260, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[XMH03] Zhiyong Xu, Rui Min, and Yiming Hu. Hieras: a dht based hierarchical p2p routing algorithm. In *Proceedings of the International Conference on Parallel Processing*, pages 187 –194, 2003.

[YPZJ05] L. Ye, Y. Peng, C. Zi, and W. Jiagao. Tcs-chord: An improved routing algorithm to chord based on the topology-aware clustering in self-organizing mode. In *Proceedings of the First International Conference on Semantics, Knowledge and Grid*, pages 25–25, Washington, DC, USA, 2005. IEEE Computer Society.

[YZLD05] M. Yang, Z. Zhang, X. Li, and Y. Dai. An empirical study of free-riding behavior in the maze p2p file-sharing system. In *Proceedings of the 4th Annual International Workshop on Peer-To-Peer Systems (IPTPS'05)*, pages 182–192, 2005.

[ZCSK07] M. A. Zaharia, A. Ch, S. Saroiu, and S. Keshav. Finding content in file-sharing networks when you can not even spell. In *Proceedings of 6th Workshop on Peer-to-Peer Systems (IPTPS07*, 2007.

[ZHS+04] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22:41–53, 2004.

[ZKJ01] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, Berkeley, CA, USA, 2001.

[ZSKT05] S. Zols, R. Schollmeier, W. Kellerer, and A. Tarlano. The hybrid chord protocol: A peer-to-peer lookup service for context-aware mobile applications. In *Proceedings of ICN*, pages 781–792, 2005.

[ZSR06] S. Zhao, D. Stutzbach, and R. Rejaie. Characterizing files in the modern gnutella network: A measurement study. In *Proceedings of the SPIE/ACM Multimedia Computing and Networking*, pages 1–13, San Jose, CA, 2006.

[ZyF02] D. Zeinalipour-yazti and T. Folias. A quantitative analysis of the gnutella network traffic. Technical report, Department of Computer Science University of California, 2002.

Publications out of the work

International Journals

1. **G. Khataniar** and D. Goswami, *Avoidance of Churn Rate Through Temporal Centralization in Chord*, Journal of Peer-to-Peer Networking and Applications, Springer, DOI:10.1007/s12083-010-0080-4, 2010.

2. **G.Khataniar** and D. Goswami, *A Hierarchical Approach to Improve Performance of Unstructured Peer-to-Peer System*, International Journal of Communication Networks and Distributed Systems (IJCNDS), Inderscience Publishers, (Accepted).

3. **G. Khataniar**, B.K. Shrivastava, S.K. Samudrala, G.B. Kumar, D. Goswami, *A Hybrid Architecture for Unstructured Peer-to-Peer Systems*, International Journal of Information Processing, ISSN 0973-8215, Volume 2, Number 3, pp.12–20, 2008.

4. **G. Khataniar** and D. Goswami,*A Hierarchical Approach to Improve Performance of p2p Systems*, ACM Transactions on Internet Technology, (under review).

Conferences

1. **G. Khataniar** and D. Goswami, *HUP: An Unstructured Hierarchical Peer-to-Peer Protocol*, In proceedings of IAENG International Conference on Internet Computing and Web Services (ICICWS), Hong-Kong, pp.671-676, 2010.

2. **G. Khataniar** and D. Goswami, *Improving Query Processing in Unstructured Peer-to-Peer System*, In proceedings of National Conference on Emerging Trends and Applications in Computer Science, Shillong, pp.10-14, 2010.

3. **G. Khataniar** and D. Goswami, *A Classical Review of Algorithms for Peer-to-Peer Systems*, In proceedings of NWDAA, Tezpur, India, pp.64-69, 2010.

4. B. Shrivastava, S. Samudrala, **G. Khataniar** and D. Goswami, *An Efficient Load Balancing Architecture for Peer to Peer Systems*, Proceedings of International Conference on Gride Computing & Applications, Las Vegas, USA, pp.18–23, 2008.

5. B. Shrivastava and **G. Khataniar** and D. Goswami, *Range Query over Structured Overlay System*, Proceedings of the International Conference on Gride Computing & Applications, Las Vegas, USA, pp.137–143, 2007.

6. B. Shrivastava, **G. Khataniar** and D. Goswami, *Performance Enhancement in Hierarchical Peer-to-Peer Systems*, Proceedings of the Communication Systems Software and Middleware, Bangalore, India, pp.1–7, 2007.

SHP OVERLAY

A.1.1 Algorithm for joining a new node in an SHP Network:

```
/*
NodeID is an integer in the range 0 - (2^m - 1). Node joins upto Initial number of groups as bootstrapping
nodes. Then the node added to the network as a temporary node. Node Status: Full-Stable Node=0,
Stable Node=1, Temporary Node=3.
*/
joinNode(NodeID, EventTime)
{
        SHPNode node = new SHPNode(NodeID)
        {
                Create a new group;
                add node to network;
                distributed join data;
                buildPeerRoutingTables(NodeID);
                modifyPeerRoutingTables(NodeID)
        }
        {
                find correct position of new node;
                insert node into network;
        }
}
```

A.1.2 Algorithm for Building Structured Routing Tables of a newly joined Node.

```
/*
MAX_BIT_NODES is a global integer representing the number of entries in a structured routing table.
MAX_BIT_NODES = (log(MAX_NODES)); MAX_NODES contains the maximum number of nodes
possible in the network.
*/
buildPeerRoutingTables(NodeID)
{
        for(i = 0; i < MAX_BIT_NODES; i++)
        {
                int key = (NodeID + 2^i)%AVERAGE_NODES;
                if (key < NodeID * AVERAGE_NODES)
                {
                        key = key + NodeID*AVERAGE_NODES;
                }
                int value = getKeySuccessor(key);
                //put (key, value) as the ith entry in the new Node's Peer routing table.
        }
}
```

getKeySuccessor(group, key)

```
{
// key is an integer value whose successor is returned by this algorithm.
        int i = 0;
        while (group.node[i] != null)
        {
                // the group object contains list of all the nodes in the group.
                If (group.node[i].ID >= key)
                {
                        return node[i].ID;
                }
                i++;
        }
        return node[0].ID;
}
```

A.1.3 Algorithm to Update Peer Routing Tables of Other Nodes on Joining of a new Node

updatePeerRoutingTables(group, newNode)

```
{
        int i = 0;
        while (group.node[i] != null)
        {
                // group object contains list of all the nodes in the group.
                For (int k = 0; k < MAX_BIT_NODES; k++)
                {
                        int key = (NodeID + 2^i)%AVERAGE_NODES;
                        if(key < NodeID * AVERAGE_NODES)
                        {
                                key = key + NodeID*AVERAGE_NODES;
                        }
                        if(group.node[i].fingerTable.get(key) == newNode.Successor)
                {
                //put (key, value) as the i^th entry in the Node[i]'s PRT
        }
        i++;
}
```

A.1.4 Algorithm for Removing an existing Node from network

removeNode(NodeID)

```
{
// The Node is first checked whether it is temporary, stable or fully-stable
If(temporary)
{
        Node is removed from the temporary node list
```

```
        Its successor and predecessor node is rearranged
}
if(stable or fullystable)
{
        Node is removed from the stable node list
        Its successor and predecessor nodes are rearranged
        PeerRoutingTable updated
        JoinDataModified
}
}
```

A.1.5 Distribution of Join data among the nodes

```
distributeData(Node)
{
        int i = 0;
        while(Node.data[i] != null)
        {
                Extract data item from the predecessor node;
                add data item to the newNode;
                remove data from the predecessor node;
                i++;
        }
}

updatePeerRoutingTables(Node)
{
        int i = 0;
        while(network.node[i] != null)
        {
                // network object contains list of all the nodes in the network.
                for(int k = 0; k < MAX_ENTRIES; k++)
                {
                        int key = (network.node[i].ID + 2^i)%MAX_NODES;
                        // MAX_NODES contains the maximum number of
                        // nodes possible in the network at a time.
                        if(network.node[i].fingerTable.get(key) == Node.ID)
                        {
                                int newValue = Node.successorID;
                                //put (key, newValue) as the i^th entry in the Node[i]'s finger table.
                        }
                        i++;
                }
        }
}
```

A.1.5 Algorithm to Search a Data Item

```
searchKey(searchKey, sourceNodeID, EventTime)
{
// search key is the key to be searched.
// The corresponding group of the source Node is evaluated
        if(data contains within the same group )
        {
                If(found in the sourceNode itsef);
                Write successful search;
                Else
                Recursive search in the other nodes under the same group;
                If(found)
                Write successful search;
                Else
                Data not present in the network;
        }
        else
        {
                The destination group is found out;
                If(found in the fully-stablenode of the group);
                Write successful search;
                Else
                Recursive search in the other nodes under the group;
                If(found)
                Write successful search;
                Else
                Data not present in the network;
        }
}
recursiveSearch(searchKey, lowerBound, nodeList, presentNodeID, recursions)
{
        if(node.data.contains(key)
        {
                print " Search Key Found";
                return key;
        }
        else
        {
                int successorID = getKeySuccessor(key).ID;
                if(successorID == node.ID)
                {
                        print " Search Key not Found";
                        return null;
                }
```

```
                else
                {
                        Node successor = getKeySuccessor(key);
//search the node's Peer routing table to find a successor of key;
Return recursiveSearch(searchKey, lowerBound, nodeList, nextNodeID, recursions);
                }
        }
}
```

A.1.6 Algorithm for a Range Search of Data Items

```
searchRange( searchKey1, searchKey2, sourceNodeID, EventTime)
{
        The group with the datarange is found out
        while(key1 <= key2)
        {
                key1 = addData(lowerbound, key1, key2, sourceNode);
                //set sourceNode to the new value according to key1
        }
}
addData(lowerBound, int key1, int key2, Node sourceNode)
{
        for(int i = 0; i <=(key2 – key1); i++)
        {
                if(node.data contains (key1+ i)
                {
                        Print “Found”
                }
        }
        return (key1+ i);
}
```

A.1.7 Algorithm for Upgradation of nodes

```
upgradeNode(EventTime)
{
//All the nodes are added to a list NodeInformation
Duration of a node = presenttime – timestamp of the node ;
If (nodeStatus == 0) // fully-stable node
{
// no need of upgradation
Continue;
}
If (nodeStataus = = 1) //stable nodes
{
        If(duration > T_STABILITY)
```

```
        {
                Increment timestamp by T_STABILITY;
                Upgrade PeerRouting tables(Node);
        }
}
If (nodeStatus == 2) //temporary node
{
        If(duration > T_AVERAGE)
        {
                Increment timestamp by T_AVERAGE;
                Add node to the stable node list;
                Build PeerRouting tables(Node);
                Distribute join data;
        }
}
```

A.1.7 Algorithm for load balancing

```
loadBalance()
{
/*
```

Nodes are graded as – HighlyLoadedNode (gradeValue >1), Normal loaded Node (0.5 < gradeValue<=1) and Lightly loaded Node (gradeValue < =.5)

```
*/
        For(each group )
        {
        While(node.gradeValue>1)
                {
                        Insert a new node between the node and its predecessor;
                        Set counter nodeAdded;
                }
        }

        While(nodeAdded > nodeRemoved)
        {
                For(eachgroup)
                {
                        if(node.gradeValue<=0.5)
                        Remove the lightlyloaded nodes ;
                        nodeRemoved++;
                        Continue;
                }
        }
}
```

CHORD OVERLAY

A.2.1 Algorithm for joining a new Node in an Chord Network:

joinNode(NodeID, EventTime)

```
{
        ChordNode node = new ChordNode(NodeID)
        {
                Set Successor Node;
                Set Predecessor Node;
                add node to network;
                distributed join data;
                buildPeerRoutingTables(NodeID);
                modifyPeerRoutingTables(NodeID);
        }
```

A.2.2 Algorithm for Building finger Tables of a newly joined Node.

buildfingerTables(NodeID)

```
{
        for(i = 0; i < MAX_BIT_NODES; i++)
        {
/*
MAX_BIT_NODES is a global integer representing the number of entries in a finger table.
MAX_BIT_NODES = (log(MAX_NODES)); MAX_NODES contains the maximum number of nodes
possible in the network.
*/
                int key = (NodeID + 2^i)%MAX_NODES;
                if (key < NodeID * MAX_NODES)
                {
                        key = key + NodeID*MAX_NODES;
                }
                int value = getKeySuccessor(key);
                //put (key, value) as the ith entry in the new Node's finger table.
        }
}
```

getKeySuccessor(group, key)

```
{
// key is an integer value whose successor is returned by this algorithm.
        int i = 0;
        while (group.node[i] != null)
        {
                // the group object contains list of all the nodes in the group.
                If (group.node[i].ID >= key)
                {
                        return node[i].ID;
```

```
                }
                i++;
        }
        return node[0].ID;
}
```

A.2.3 Algorithm to Update finger Tables of Other Nodes on Joining of a new Node
updatefingerTables(newNode)

```
{
        int i = 0;
        while (group.node[i] != null)
        {
                // group object contains list of all the nodes in the group.
                For (int k = 0; k < MAX_BIT_NODES; k++)
                {
                        int key = (NodeID + 2^i)%MAX_NODES;
                        if(key < NodeID * MAX_NODES)
                        {
                                key = key + NodeID*MAX_NODES;
                        }
                        if(group.node[i].fingerTable.get(key) == newNode.Successor)
                {
                //put (key, value) as the i^th entry in the Node[i]'s finger table.
        }
        i++;
}
```

A.2.4 Algorithm for Removing an existing Node from network removeNode(NodeID, EventTime)

```
{
//Position of the node in the chord list is found out
        Node is removed from the chord node list
        Its successor and predecessor nodes are rearranged
        modifyLeaveFingerTables(ID, successorIndex);
        distributeLeaveData(ID, successorIndex);
}
```

A.2.5 Distribution of Join data among the nodes
distributeData(Node)

```
{
        int i = 0;
        while(Node.data[i] != null)
        {
                Extract data item from the predecessor node;
```

```
                add data item to the newNode;
                remove data from the predecessor node;
                i++;
        }
}

updatefingerTables(Node)
{
        int i = 0;
        while(network.node[i] != null)
        {
                // network object contains list of all the nodes in the network.
                for(int k = 0; k < MAX_BIT_NODES; k++)
                {
                        int key = (network.node[i].ID + 2^i)%MAX_NODES;
                        // MAX_NODES contains the maximum number of
                        // nodes possible in the network at a time.
                        if(network.node[i].fingerTable.get(key) == Node.ID)
                        {
                                int newValue = Node.successorID;
                                //put (key, newValue) as the i^th entry in the Node[i]'s finger table.
                        }
                        i++;
                }
        }
}
```

A.2.5 Algorithm to Search a Data Item

```
searchKey(searchKey, sourceNodeID, EventTime)
{
        // search key is the key to be searched.
        {
                If(found in the sourceNode itsef);
                {
                Write successful search;
                }
                Else
                {
                        recursiveSearch(searchKey, sourceNodeID, 1, 0);
                }
        }
}
recursiveSearch(searchKey, lowerBound, nodeList, presentNodeID, recursions)
{
```

```
        if(node.data.contains(key)
        {
                print " Search Key Found";
                return key;
        }
        else
        {
                int successorID = getKeySuccessor(key).ID;
                if(successorID == node.ID)
                {
                        print " Search Key not Found";
                        return null;
                }
                else
                {
                        Node successor = getKeySuccessor(key);
//search the node's Peer routing table to find a successor of key;
Return recursiveSearch(searchKey, lowerBound, nodeList, nextNodeID, recursions);
                }
        }
}
```

A.6 Algorithm for a Range Search of Data Items

```
searchRange( searchKey1, searchKey2, sourceNodeID, EventTime)
{
        for(int i = searckKey1; i <= searchKey2; i++)
        {
                searchKey(i, sourceNodeID);
                //serchKey is same as point search in chord

        }
}
```

Appendix A : shp.txt

The Network Simulation Parameters are :

1. T_STABILITY	**: 2000**
2. T_AVERAGE	**: 100**
3. Number of Nodes in each Group	**: 256**
4. Number of Groups	**: 8**
5. Key Interval	**: 10**
6. Maximum number of nodes	**: 2048**
7. Total number of Events	**: 50**

The Events occurring in the Network are :

Time : 0	**Event : Join of a new Node.**
Time : 2	**Event : Join of a new Node.**
Time : 4	**Event : Join of a new Node.**
Time : 6	**Event : Join of a new Node.**
Time : 8	**Event : Join of a new Node.**
Time : 16	**Event : Join of a new Node.**

Appendix A : shp.txt

Time : 19 Event : Join of a new Node.

Time : 19 Event : Join of a new Node.

Time : 20 Event : Join of a new Node.

Time : 20 Event : Search of a data Item. Search Key : 1530

Time : 20 Event : Join of a new Node.

Time : 20 Event : Search of a data Item. Search Key : 268

Time : 22 Event : Search of a data Item. Search Key : 1977

Time : 24 Event : Search of a data Item. Search Key : 1786

Time : 26 Event : Search of a data Item. Search Key : 781

Time : 27 Event : Join of a new Node.

Time : 27 Event : Search of a data Item. Search Key : 1453

Time : 27 Event : Search of a data Item. Search Key : 1086

Time : 27 Event : Search of a data Item. Search Key : 1960

Appendix A : shp.txt

Time : 29 Event : Search of a data Item. Search Key : 979

Time : 29 Event : Join of a new Node.

Time : 30 Event : Range Search of data Items.

Time : 30 Event : Search of a data Item. Search Key : 158

Time : 32 Event : Join of a new Node.

Time : 34 Event : Join of a new Node.

Time : 35 Event : Search of a data Item. Search Key : 681

Time : 37 Event : Join of a new Node.

Time : 39 Event : Search of a data Item. Search Key : 1449

Time : 40 Event : Search of a data Item. Search Key : 1915

Time : 40 Event : Search of a data Item. Search Key : 565

Time : 40 Event : Search of a data Item. Search Key : 1154

Time : 40 Event : Search of a data Item. Search Key : 540

Appendix A : shp.txt

Time : 40 Event : Search of a data Item. Search Key : 1551

Time : 42 Event : Search of a data Item. Search Key : 765

Time : 42 Event : Join of a new Node.

Time : 42 Event : Range Search of data Items.

Time : 42 Event : Join of a new Node.

Time : 43 Event : Search of a data Item. Search Key : 252

Time : 45 Event : Search of a data Item. Search Key : 110

Time : 46 Event : Search of a data Item. Search Key : 1631

Time : 46 Event : Search of a data Item. Search Key : 1192

Time : 47 Event : Range Search of data Items.

Time : 47 Event : Range Search of data Items.

Time : 48 Event : Search of a data Item. Search Key : 335

Time : 49 Event : Search of a data Item. Search Key : 1513

Appendix A : shp.txt

Time : 51 Event : Search of a data Item. Search Key : 12

Time : 51 Event : Search of a data Item. Search Key : 1038

Time : 52 Event : Search of a data Item. Search Key : 1759

Time : 53 Event : Join of a new Node.

Time : 54 Event : Join of a new Node.

TEMPORALSIM

B.1 Algorithm for joining a new Node in an Chord Network:

joinNode(NodeID, EventTime)

```
{
// NodeID is an integer in the range 0 - (2^m - 1)
        ChordNode node = new ChordNode(NodeID)
        if( I < MAX_NODE_BITS)
        //Bootstrapping nodes – node joins as fully-stable
        {
                Set Successor Node;
                Set Predecessor Node;
                add node to network;
                distributed join data;
                buildPeerRoutingTables(NodeID);
                modifyPeerRoutingTables(NodeID);
                i++;
        }
        Else
        {
        Find the closest stable node ;
        Join the new node through the stable node;
        }
```

B.2 Algorithm for Building finger Tables of a newly joined Node.

buildfingerTables(NodeID)

```
{
        for(i = 0; i < MAX_BIT_NODES; i++)
        {
                /*
                MAX_BIT_NODES is a global integer representing
                the number of entries in a finger table.
                MAX_BIT_NODES = (log(MAX_NODES));
                MAX_NODES contains the maximum number of
                nodes possible in the network.
                */
                int key = (NodeID + 2^i)%MAX_NODES;
                if (key < NodeID * MAX_NODES)
                {
                        key = key + NodeID*MAX_NODES;
                }
                int value = getKeySuccessor(key);
                //put (key, value) as the ith entry in the new Node's finger table.
        }
}
```

```
getKeySuccessor(group, key)
{
// key is an integer value whose successor is returned by this  algorithm.
        int i = 0;
        while (group.node[i] != null)
        {
                // the group object contains list of all the nodes in the group.
                If (group.node[i].ID >= key)
                {
                        return node[i].ID;
                }
                i++;
        }
        return node[0].ID;
}
```

B.3 Algorithm to Update finger Tables of Other Nodes on Joining of a new Node

```
 // upgradation is done after every T_average
//Counter is incremented after every T_average
updatefingerTables(newNode , counter)
{
        int i = 0;
        while (counter < MAX_NODE_BITS)
        {
                // group object contains list of all the nodes in the group.
                For (int k = 0; k < counter; k++)
                {
                        int key = (NodeID + 2^i)%MAX_NODES;
                        if(key < NodeID * MAX_NODES)
                        {
                                key = key + NodeID*MAX_NODES;
                        }
                        if(group.node[i].fingerTable.get(key) == newNode.Successor)
                {
                //put (key, value) as the i^th entry in the Node[i]'s finger table.
                //node is a quasiStable node;
        }

        While(counter = = MAX_NODE_BITS)
        {
         // distribute join data; node upgraded to fullystable node;
        }
        i++;
```

```
}
```

B.4 Algorithm for Removing an existing Node from network removeNode(NodeID, EventTime)

```
{
//Position of the node in the chord list is found out
        When(node.status = 0) // temporaryNode
        {
         Remove the node from the temporarynode list;
        Update TRT of the stable node;
        }
        While(node.status = = 1) // quasistable node
        {
                Node removed from the chord list;
                Update finger table entries of other nodes upto counter ;
        }
        While (node.status = = 2) // fully-stable node
        {
                Node removed from the chord node list;
                Update all finger table entries;
                Distribute leave data;
        }

}
```

B.5 Distribution of Join data among the nodes

distributeData(Node)

```
{
        int i = 0;
        while(Node.data[i] != null)
        {
                Extract data item from the predecessor node;
                add data item to the newNode;
                remove data from the predecessor node;
                i++;
        }
}
```

updatefingerTables(Node)

```
{
        int i = 0;
        while(network.node[i] != null)
        {
                // network object contains list of all the nodes in the network.
                for(int k = 0; k < MAX_BIT_NODES; k++)
                {
```

```
                        int key = (network.node[i].ID + 2^i)%MAX_NODES;
                        // MAX_NODES contains the maximum number of
                        // nodes possible in the network at a time.
                        if(network.node[i].fingerTable.get(key) == Node.ID)
                        {
                                int newValue = Node.successorID;
                                //put (key, newValue) as the i^th entry in the Node[i]'s finger table.
                        }
                        i++;
                }
        }
}
```

B.5 Algorithm to Search a Data Item

```
searchKey(searchKey, sourceNodeID, EventTime)
{

        // search key is the key to be searched.
        // search the key only in the fullystable node in the system (chord node list)
        {
                If(found in the sourceNode itsef);
                {
                Write successful search;
                }
                Else
                {
                        recursiveSearch(searchKey, sourceNodeID, 1, 0);
                }
        }
}
recursiveSearch(searchKey, lowerBound, nodeList, presentNodeID, recursions)
{
        if(node.data.contains(key)
        {
                print " Search Key Found";
                return key;
        }
        else
        {
                int successorID = getKeySuccessor(key).ID;
                if(successorID == node.ID)
                {
                        print " Search Key not Found";
                        return null;
```

```
                }
                else
                {
                        Node successor = getKeySuccessor(key);
                        //search the node's Peer routing table to find a successor of key;
Return recursiveSearch(searchKey, lowerBound, nodeList, nextNodeID, recursions);
                }
        }
}
```

Appendix B : tcp.txt

The Network Simulation Parameters are :

1. T_AVERAGE	**: 12**
2. Maximum number of nodes	**: 4096**
3. Total number of Events	**: 50**

The Events occurring in the Network are :

Time : 0	**Event : Join of a new Node.**
Time : 2	**Event : Join of a new Node.**
Time : 2	**Event : Join of a new Node.**
Time : 3	**Event : Join of a new Node.**
Time : 5	**Event : Join of a new Node.**
Time : 7	**Event : Join of a new Node.**
Time : 8	**Event : Join of a new Node.**
Time : 14	**Event : Join of a new Node.**

Appendix B : tcp.txt

Time : 16 **Event : Join of a new Node.**

Time : 16 **Event : Join of a new Node.**

Time : 18 **Event : Join of a new Node.**

Time : 18 **Event : Join of a new Node.**

Time : 19 **Event : Search of a data Item.** **Search Key : 3683**

Time : 20 **Event : Join of a new Node.**

Time : 20 **Event : Join of a new Node.**

Time : 20 **Event : Join of a new Node.**

Time : 22 **Event : Join of a new Node.**

Time : 23 **Event : Join of a new Node.**

Time : 24 **Event : Join of a new Node.**

Time : 25 **Event : Search of a data Item.** **Search Key : 3302**

Time : 26 **Event : Search of a data Item.** **Search Key : 1051**

Appendix B : tcp.txt

Time : 27 **Event : Join of a new Node.**

Time : 28 **Event : Search of a data Item.** **Search Key : 903**

Time : 29 **Event : Search of a data Item.** **Search Key : 2100**

Time : 29 **Event : Join of a new Node.**

Time : 31 **Event : Join of a new Node.**

Time : 32 **Event : Join of a new Node.**

Time : 33 **Event : Search of a data Item.** **Search Key : 2624**

Time : 35 **Event : Join of a new Node.**

Time : 35 **Event : Join of a new Node.**

Time : 35 **Event : Join of a new Node.**

Time : 35 **Event : Join of a new Node.**

Time : 36 **Event : Join of a new Node.**

Time : 37 **Event : Search of a data Item.** **Search Key : 3623**

Appendix B : tcp.txt

Time : 39 Event : Search of a data Item. Search Key : 3472

Time : 39 Event : Join of a new Node.

Time : 41 Event : Join of a new Node.

Time : 43 Event : Search of a data Item. Search Key : 1220

Time : 43 Event : Join of a new Node.

Time : 44 Event : Search of a data Item. Search Key : 3241

Time : 46 Event : Join of a new Node.

Time : 46 Event : Join of a new Node.

Time : 46 Event : Leaving of an existing Node.

Time : 46 Event : Join of a new Node.

Time : 48 Event : Join of a new Node.

Time : 48 Event : Join of a new Node.

Time : 49 Event : Join of a new Node.

Appendix B : tcp.txt

Time : 50 Event : Join of a new Node.

Time : 52 Event : Join of a new Node.

Time : 54 Event : Join of a new Node.