P-Skip Graph: An Efficient Data Structure for Peer-to-Peer Network

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering

Submitted By
Amrinderpreet Singh
(Roll No. 801131031)

Under the supervision of:
Dr. Shalini Batra
Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

July 2013

Certificate

I hereby certify that the work which is being presented in the thesis entitled, "*P-Skip Graph : An Efficient Data Structure for Peer-to-Peer Network*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Shalini Batra* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Amrinder Preet Singh
(Amrinderpreet Singh)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Shalini Batra)
Assistant Professor
Computer Science and Engineering Department
Thapar University, Patiala

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

No volume of words is enough to express my gratitude towards my guide **Dr. Shalini Batra**, Department of Computer Science & Engineering, Thapar University, Patiala, who has been very concerned and has aided for all the materials essentials for the preparation of this thesis report. She has helped me to explore this vast topic in an organized manner and provided me all the ideas on how to work towards a research-oriented venture.

I am also thankful to **Dr. Maninder Singh**, Head of Computer Science & Engineering Department and **Mr. Karun Verma**, P.G. Coordinator, for the motivation and inspiration that triggered me for the thesis work.

I would also like to thank the staff members and my colleagues who were always there at the need of hour and provided with all the help and facilities, which I required, for the completion of my thesis work.

Most importantly, I would like to thank my parents and the almighty for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.

Amrinderpreet Singh

Amrinderpreet Singh

(801131031)

Abstract

With the growth of the Internet and the emergence of the trend of moving from desktop computing to client-server computing and finally to distributed computing, a class of distributed network viz. peer-to-peer network has gained huge popularity. Peer-to-peer network displays interesting characteristics of fast queries, updation, deletion, fault-tolerance and much more while lacking any central authority. With a view of changing trend in mind, the major concern of computer experts is to develop an appropriate data structure than can efficiently handle all the desired characteristics of the underlying network. Adjacency Matrix Skip-Webs, Skip-Nets, Skip-List, Distributed Hash Table, and many more data structures form the candidature for peer-to-peer networks, of which, Skip-Graph (evolved version of skip-list) displays the best characteristics.

Researchers keep on optimizing the existing data structures according to the need of the area where the data structure is to be applied. Many optimization techniques have been applied to adapt a data structure to a particular scenario and usage of Skip-Graphs is one such option. Skip-Graph help to search and locate a node in a peer-to-peer network efficiently with time complexity being *O(log n)*. However when a hotspot node is searched and queried again and again, the Skip-Graph does not learn or adapt to the situation and still searches traditionally with *O(log n)* complexity.

The main focus of the thesis is to modify traditional Skip-Graph data structure so that it can automatically learn and adapt to the environment in peer-to-peer network. The aim is to use to top most level of the Skip-Graph node to establish a link to the most frequently searched node so that satisfying a trend of similar queries becomes as easy as *O(1)*.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

The word distributed in terms such as "distributed system", "distributed programming", and "distributed algorithm" originally referred to computer networks where individual computers were physically distributed within some geographical area [1]. The use of concurrent processes that communicate by message-passing has its roots in operating system architectures studied in the 1960s [2]. The first widespread distributed systems were local-area networks such as Ethernet that was invented in the 1970s [3]. ARPANET, the predecessor of the Internet, was introduced in the late 1960s, and ARPANET e-mail was invented in the early 1970s. E-mail became the most successful application of ARPANET [3], and it is probably the earliest example of a large-scale distributed application. In addition to ARPANET, and its successor, the Internet, other early worldwide computer networks included Usenet and Fido Net from 1980s, both of which were used to support distributed discussion systems.

The terms are nowadays used in a much wider sense, even referring to autonomous processes that run on the same physical computer and interact with each other by message passing [2]. There is no single definition of a distributed system, however there are a few defining properties as follows:

- There are several autonomous computational entities, each of which has its own local memory [4]
- The entities communicate with each other by message passing [2]
- There is no central control of the distributed system [1,2].

The computational entities are called "computers" or "nodes". A distributed system may have a common goal, such as solving a large computational problem [5,6] or each computer may have its own user with individual needs, and the purpose of the distributed system is to coordinate the use of shared resources or provide communication services to the users .

The study of distributed computing became its own branch of computer science in the late 1970s and early 1980s. The first conference in the field, Symposium on Principles of Distributed Computing (PODC), dates back to 1982, and its European

counterpart International Symposium on Distributed Computing (DISC) was first held in 1985. Distributed systems have become very popular for internet applications in a very short period of time.

1.2 A Brief introduction to types of distributed systems

Various hardware and software architectures are used for distributed computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed onto a circuit board or made up of loosely coupled devices and cables. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system.

Distributed systems typically falls into one of several basic architectures [7] or categories:

- **Client–server:** In this type of architecture there is a dedicated machine which provides service to the clients. This machine is known as Server since it has to serve the client requests. A smart client code contacts the server for data then formats and displays it to the user. Input at the client is committed back to the server when it represents a permanent change. The clients in this type of architecture merely exploit the server power. This type of architecture depicts a single point of failure.
- **3-tier architecture:** Three tier systems move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Clients also known as thin-clients only need to know how the present the information received from the server to the user. The main logic resides with the middle tier therefore is easy to change, modify or delete. Most web applications are 3-Tier.
- ***n*-tier architecture**: *n*-tier refers typically to web applications which further forward their requests to other enterprise services.
- **Highly coupled (clustered):** This architecture contains cluster of machines that closely work together, running a shared process in parallel. The task is subdivided in parts that are made individually by each one and then put back together to make the final result.
- **Space based:** Space-based architecture refers to an infrastructure that creates the illusion (virtualization) of one single address-space. Data are transparently replicated according to application needs. Decoupling in time, space and reference is achieved.
- **Peer-to-peer:** This is an architecture where there is no special machine or machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as

clients and servers. Hence, the name , peer-to-peer. This architecture leverages the powers of all the systems that participate in the network and hence are have gain a lot of importance in a very short period of time.

1.3 Peer to Peer Network

Distributed peer to peer networks present a decentralized, distributed method of storing large data sets. Information is stored at the hosts and queries are performed by sending messages between the hosts so as to ultimately identify the host(s) that store(s) the requested information. They are distributed systems without any central authority that are used for efficient location of shared resources. Some of the desirable features of peer-to-peer network are decentralization, scalability, fault-tolerance, self-stabilization , data-availability, load-balancing, efficient and complex query searching [8].

Peer-to-peer computing is another term which is often used in literature [9]. This is a general term given to the process of sharing resources and services between computers by direct exchange between the systems. These resources and services include the exchange of information, processing cycles, cache storage, and disk storage for files. P2P computing takes advantage of existing computing power, computer storage and networking connectivity, allowing users to leverage their collective power to the 'benefit' of all. Figure 1.1 displays a typical server-based and peer-to-peer network.

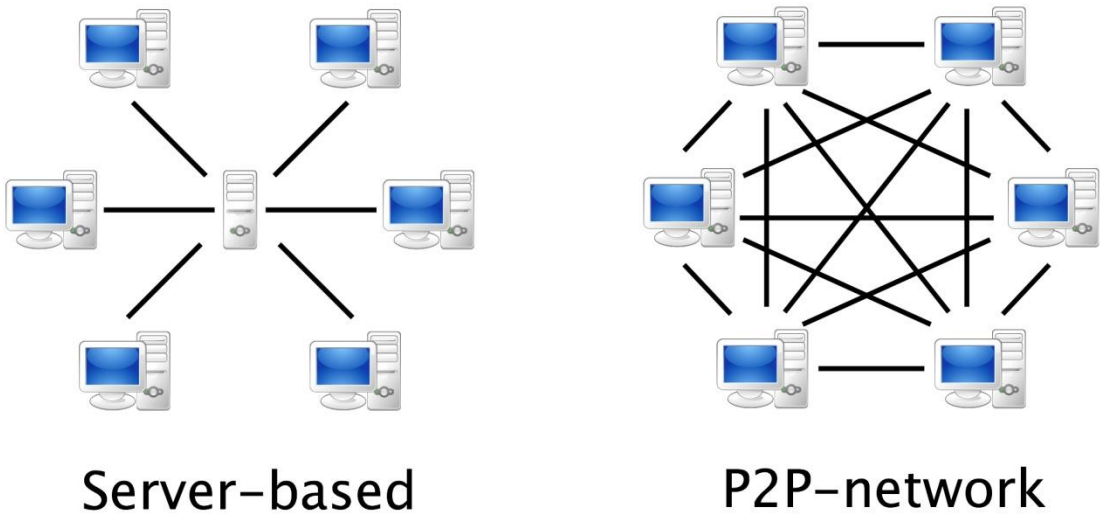


Figure 1.1: A typical Server-Based Network(left) and peer-to-peer network(right). [9]

Examples of peer-to-peer network include File sharing (Napster, Gnutella, Kazaa) , Multiplayer games (Unreal Tournament, DOOM), Collaborative applications (ICQ, shared whiteboard), Distributed computation (Seti@home) , Ad-hoc networks , biological - networks (like neurons) , food web and neural networks [10], metabolic pathways, computation networks like Grid-computing [11]. Distributed system (especially peer-to-peer) focuses on real time scenarios of the world where the resources are distributed geographically among the nodes and not all the nodes know every aspect of the network it is a part of. In such a network the node only knows few of the other nodes. These are the nodes that are the candidate for introducing the new node to the network. Such nodes are also the only means through which the concerned node can query about the resources being shared among the network nodes, which node has which resource which node to connect , to get a particular resource. Such a scenario is often seen in typical torrent applications for file-sharing.

1.4 Simplified P2P Architecture and Characteristics

The key feature of a peer-to-peer network is that this type of network lacks a central authority or a server to control the other participating nodes, reducing the probability of single point of failure. Furthermore peer to peer system takes advantage of the computation power of each and every node and hence balances the load to all the nodes of the system.

Since all the nodes are both client and server in P2P network all can provide and consume data. They also act as routers for forwarding the message from one node to the other in order to send the message to its final destination. Any node can initiate a connection as there is no centralized data source. Nodes or the systems have the computation capabilities which they can offer to the whole system while collaborating collectively. The nodes need not be in the same geographic area. The nodes may be distributed across the globe and may be connected to the peer network over the internet. Figure 1.2 shows a typical p2p network over the internet.

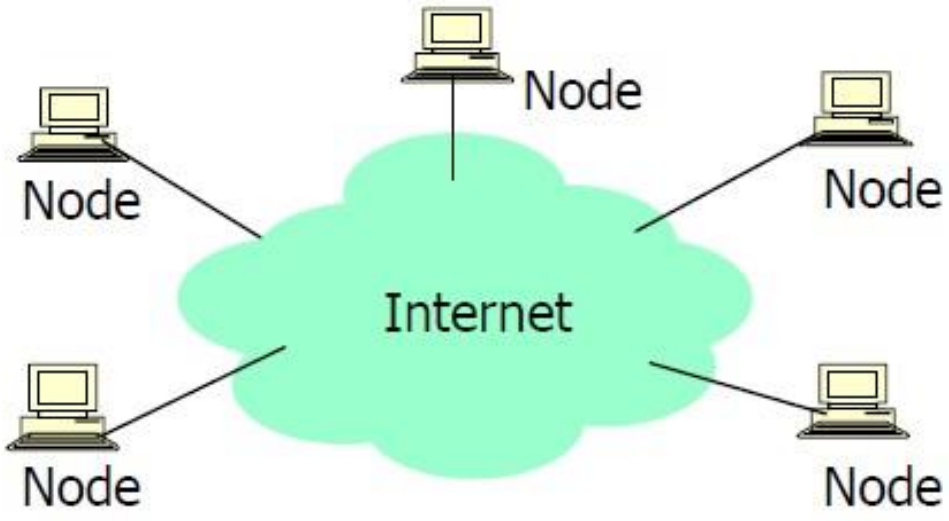


Figure 1.2: A typical peer-to-peer network over the internet [12]

Some of the characteristics of peer-to-peer network are :

- Clients are also servers and routers. Nodes contribute content, storage, memory, CPU, *etc.*
- Nodes are autonomous (no administrative authority). They all share the same algorithm with which they work and collaborate with each other.
- The network is dynamic. The nodes enter or leave the network frequently.
- Nodes collaborate directly with each other (not through well-known servers).

P2P needs to address some important issues like small nodes size, fault tolerance, fast queries and updates and support for ordered data before a data structure is selected for its implementation.

1.5 Graphs and p2p networks

Since any network topology can be depicted in the form of generalized graphs, Peer-to-peer network is diagramatically represented as vertices(nodes) and edges(links between the nodes).

Figure 1.3 shows a 6 node network where each vertex represents the nodes that participate in the network and the links represents the known peers (*i.e.* the nodes that are neighbours).

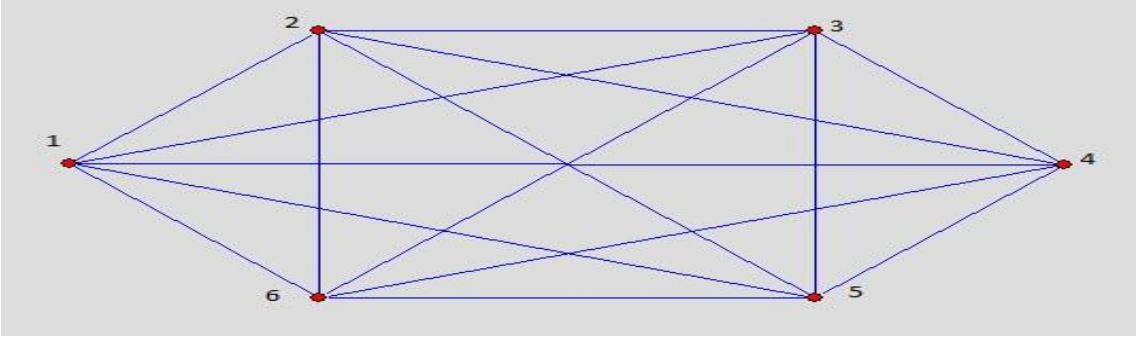
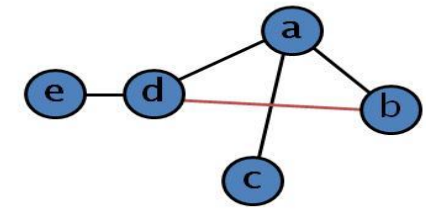


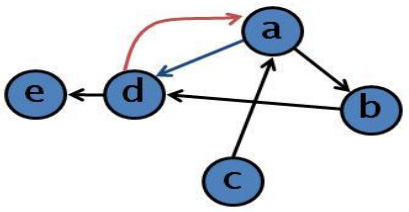
Figure 1.3: Graphical representation of 6 node network

1.5.1 Adjacency Matrix for p2p

Typically, graphs are normally stored in the form of adjacency matrix, adjacency list etc. Such representations are good for a central server that knows each and every member of the network.



	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	0	0
d	1	1	0	0	1
e	0	0	0	1	0



	a	b	c	d	e
a	0	1	0	1	0
b	0	0	0	1	0
c	1	0	0	0	0
d	1	0	0	0	1
e	0	0	0	0	0

Figure 1.4: Adjacency Matrices for graphs depicted in the left of figure. [8]

However in case of p2p since all the nodes only need to store the information of some of its neighbours, storing such matrix on every node would mean wastage of space. Moreover the adjacency matrix is n x n representation of the vertices and links. Therefore as the number of nodes in the network grow, the space complexity grows tremendously by $O(n^2)$. Moreover adjacency matrix is not a successful distributed data structure since updation, deletion, would be too cumbersome for each and every node.

1.5.2 Skiplist for p2p

One of the data structure which can be used for efficient storage of graph information of P2P network is Skip list [13]. A skip list is built in layers where the bottom layer is an ordinary ordered linked list. Each higher layer acts as an "express lane" for the lists below, where an element in layer *i* appears in layer *i*+1 with some fixed probability *p* (two commonly used values for *p* are 1/2 or 1/4). On average, each element appears in 1/(1-*p*) lists, and the tallest element (usually a special head element at the front of the skip list) in $\log_{1/p}(n)$ lists.

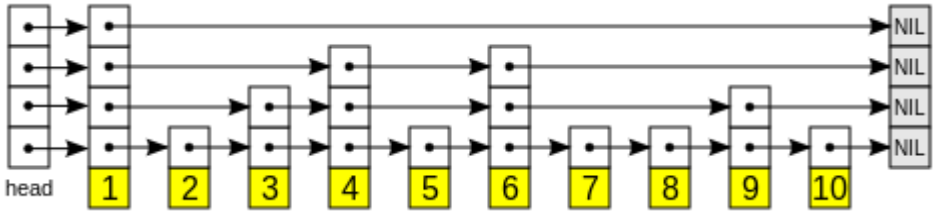


Figure 1.5: Pictorial representation of skiplist [13]

A search for a target element begins at the head element in the top list, and proceeds horizontally until the current element is greater than or equal to the target. If the current element is equal to the target, it has been found. If the current element is greater than the target, or the search reaches the end of the linked list, the procedure is repeated after returning to the previous element and dropping down vertically to the next lower list. The expected number of steps in each linked list is at most 1/*p*, which can be seen by tracing the search path backwards from the target until reaching an element that appears in the next higher list or reaching the beginning of the current list. Therefore, the total *expected* cost of a search is $(\log_{1/p}(n))/p$ which is $O(\log n)$ when *p* is a constant [13]. By choosing different values of *p*, it is possible to trade search costs against storage costs.

However skiplists also pose a problem of single point of failure. Since every search has to begin from the head. If the header node is down, there would be a complete network failure. Moreover, this data structure is not fully distributed since any node that needs to query a certain item must know the starting node. Hence every operation beginning from the same node reduces the network efficiency (since every time the search would begin from the beginning) and the network does tolerate fault.

1.5.3 SkipGraphs for peer-to-peer network

Skip graphs [14] are a kind of distributed data structure based on skip lists. They have the full functionality of a balanced tree in a distributed system. Skip graphs are mostly used in searching peer-to-peer networks. As they provide the ability to query by key ordering, they improve other search tools based on the hash table functionality only. In contrast to skip lists and other tree data structures, they are very resilient and can tolerate a large fraction of node fails. Also, constructing, inserting, searching and repairing a skip graph that was disturbed by failing nodes can be done by simple and straightforward algorithm.

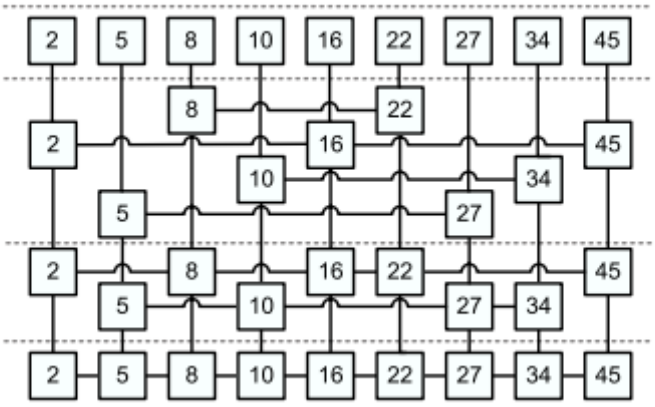


Figure 1.6: A skip graph with dashed line showing different levels [15]

Since almost all the important issues viz. fault tolerance, small sized nodes, fast queries and ordered data are addressed by skip graph, the thesis takes up skip graph as a base for p2p network and tries to improve certain aspects addressed in the literature review. Our objective

is to utilize the space reserved by the top level of the skip graph nodes for probabilistically linking nodes to each other to further improve the searching time in p2p networks.

1.6 Structure of the Thesis

The rest of the thesis is organized in the following order:

Chapter 2 – Literature Review: This chapter surveys various techniques and data structures used in the literature for p2p networks.

Chapter 3 - Problem Statement: This chapter states the problem and provides the methodology used to solve it.

Chapter 4 - Design of P-Skip Graph: This chapter gives a detailed information about the P-skipgraphs (modified from traditional skipgraphs) and how it improves the search time in p2p

Chapter 5 - Implementation Details: It includes the experiments performed and the results achieved.

Chapter 6 - Conclusion and Future Scope: It concludes the thesis and provides suggestions for future work.

Thesis concludes with references and list of publications.

Chapter 2
Literature Review

Since the advent of computers, man has always wanted to fully utilize the power of the computer and to take to the extremes. Technology however paved the way for faster and faster processors and ultimately led mankind to develop superfast computers that could performs trillions of calculations at a very high speed.

However with the widespread of Personal Computers with low-power CPUs it was not possible to take advantage of every household computer as a super-computer. While there was research going on in the field of enhancing the power of a single computer with multiple powerful CPUs , 1960 saw the beginning of trivial distributed computing with the use of concurrent processes that communicate by message-passing ,having its roots in operating system architectures [2].

The terms "concurrent computing", "parallel computing", and "distributed computing" have a lot of overlap, and no clear distinction exists between them [16,17]. The same system may be characterised both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel [18]. Parallel computing may be seen as a particular tightly coupled form of distributed computing [17] and distributed computing may be seen as a loosely coupled form of parallel computing. Nevertheless, it is possible to roughly classify concurrent systems as "parallel" or "distributed" using the following criteria:

- In parallel computing, all processors may have access to a shared memory to exchange information between processors.[19]
- In distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors. [19,20]

With the benefits of distributed computing being known to the world, various architectures began to be developed for using normal personal computers to participate in a network to take leverage of the benefits of distributed systems. All kinds of architectures were developed viz. client-server, 3-tier, n-tier, clustered, space-based and peer-to-peer. The growth of the internet was due to the successful deployment of client-server architecture. WWW, HTTP, etc all were popularized because of the client-server architecture. However various limitations of the traditional client-server architecture led to the rise of trivial peer-to-peer architecture.

Peer-to-peer distributed systems became popular in the Internet Applications in a short period of time. Survey reveals a slew of desirable features for a peer-to-peer network such as decentralization , scalability, fault-tolerance, self-stabilization, data-availability, load-balancing, dynamic addition and deletion of nodes, efficient and complex query searching [14,15] incorporating geography in searches and exploiting spatial as well as temporal locality of the resources.

Optimization today is a basic research tool in all areas of engineering, medicine, and the sciences. Storage optimization has been expanding in all directions at an astonishing rate during the last few decades. New algorithmic and theoretical techniques have been developed, the diffusion into other disciplines has proceeded at a rapid pace, and knowledge of all aspects of the field has grown even more profound. At the same time, one of the most striking trends in optimization is the constantly increasing emphasis on the interdisciplinary nature of the field. The decision-making tools based on optimization procedures are successfully applied in a wide range of practical problems arising in virtually any sphere of human activity. Optimization in memory usage and computational speed is always main concern of research in every field.

Various data structure depending on the nature of the problem, formulate different techniques for typical optimization problems. Data structures as Skip list[13], Skip Graphs[14], DHT [12] deals with optimization problems, in which the objective and constraints can be formulated using only functions that are linear with respect to the decision variables. In nonlinear optimization, one deals with optimizing a nonlinear function over a feasible domain described by a set of, in general, nonlinear functions. The pioneering works on p2p networks by J. Aspens generated a great deal of research enthusiasm in the area of various data structures, resulting in a number of new techniques for solving large-scale dynamic p2p network [14].

Recent systems like CAN [21], Chord [22], Pastry [23], Tapestry [24] and Viceroy [25] use a Distributed Hash Table (DHT) approach to overcome scalability problem. To ensure scalability they hash the key of a resource to determine which node it will be stored at and balance out the load on the nodes in the network. The main operation in these systems is to retrieve the identity of the node that stores the resource, from any other node in the network. To this end there is an overlay graph in which the location of the node and the resources is determined by the hashed values of their identities and keys respectively. Resource location using overlay graph is done in these various systems by using different routing algorithms. Pastry and Tapestry use Plaxton's algorithm, which is based on hypercube routing [23,24].

The message is forwarded deterministically to a neighbour whose identifier is one digit closer to the target identifier. CAN partitions a d-dimensional space into zones that are owned by the nodes which store keys mapped to their zone. Routing is done by greedily forwarding messages to the neighbour closest to the target zone [21]. Chord maps nodes and resources to identities of m bits placed around a modulo 2^m identifier circle and does greedy routing to the farthest possible node stored in the routing table. Most of these systems use O(log n) space and time for routing and O($\log^2$ n) time for node insertion [22].

The initial systems such as Napster [26], Gnutella [27], and Freenet [28] did not support most of the features and clearly were unscalable either due to the use of central server (Napster) or due to high message complexity from performing searches by flooding the network (Gnutella).

In many optimization problems such as p2p network linking and storage, as well as other applications, the input data, such as like node entry, failure, update, *etc*., are stochastic. In addition to the difficulties encountered in deterministic optimization problems, the stochastic problems introduce the additional challenge of dealing with uncertainties. To handle such problems, one needs to utilize probabilistic methods alongside optimization techniques. This led to the development of an advanced data structure called Skip Graph [14], whose objective is to provide tools to help design and control stochastic systems with the goal of optimizing their performance. Here main concern in optimization is dynamic allocation of memory, fast search results with minimum computational cycles.

With the enormous growth of Internet the world has become closer and faster. The major concern for computer experts is how to store such enormous amount of data especially in form of graphs. Further, data structure used for storage of such type of data should provide efficient format for fast retrieval of data as and when required. Although adjacency matrix is an effective technique to represent a graph having few or large number of nodes and vertices but when it comes to distributed nodes and probabilistic entry and failure of nodes, adjacency matrix cannot do this.

Some of the systems discussed earlier in the section have good search complexity, but are very sensitive to node failures and thus their performance degrade with random node failures. Interesting variants of skip graphs like skip webs [29], skip nets [14] and rainbow skip-graphs [15] have also been studied in the past due to the good results of skip graphs in p2p.

In this thesis we provides a special kind of data structure, P-Skip Graph which is a further modification of trivial Skip Graph and can be efficiently used for searching in p2p network

by taking into account the stochastic real-time search queries and adjusting the data structure for efficient future searches.

Skip List

A skip list is an ordered data structure based on a succession of linked lists with geometrically decreasing numbers of items [13]. The deterministic versions of skip list have guaranteed properties whereas randomized skip lists only offer high probability performance. This height (Hn) is the maximum length of a search path for any key from the top of the skip list.

Skip Graph

Skip Graphs as has been discussed in Section 1.4.3 is an extension of skip list., The whole data structure can be distributed among a large number of nodes, and the structure provides good load balancing and fault tolerance properties. Skip graphs have functionality best suitable for P2P distributed environments [6] and they perform queries based on key ordering, improving on existing search tools that provide only hash table functionality [14]. skip graphs are highly resilient, tolerating a large fraction of failed nodes without losing connectivity [14]. In addition, constructing, inserting new elements, searching a skip graph and detecting and repairing errors in the data structure introduced by node failures can be done using simple and straightforward algorithms.

Due to the excellent scalability, fault tolerant and fast retrieval offered by Skip Graph they are quite popular in various application areas viz .

- Dynamic Overlay Network (a logical network formed over a wired or wireless routing network)
- Multithreaded, multiprocessor or distributed systems, where many entities access the shared pool concurrently [2,4]
- Peer-to-Peer network [15](where all the nodes are interconnected without any central authority)
- Cloud Computing [30] for authenticity and persistence in data-centric applications.

Features offered by Skip Graphs

Efficient Search

The average search in skip graph involves only O(log n) nodes that most searches succeed as long as the proportion of failed nodes is substantially less than O(logn) [12,14,15] .

Efficient Updation and Deletion

Since Skip Graphs are essentially linked lists therefore insertion and deletion can be easily linearized, since only level 0 is there to hold the nodes. The rest of the levels are merely there to provide express ways for efficient searches.

Tolerance to failures

In [14, 15] authors prove experimentally that the skip graph does not fail completely unless 2/3 of the node links fail , thereby making the data structure tolerant to failure. It can also be observed visually that there exist numerous links between the nodes hence failure of a few nodes would not halt the network.

After analyzing various research proposals in the area of p2p networks, it was observed that among all the data structures used so far Skip Graph emerges out to be one of the best candidate for p2p network with all its interesting features where insertion, deletion and search all can be done in amortized O(log n) times while being highly tolerant to node failures and maintaining small node sizes.

However it has been observed throughout the literature that if same query is performed from same node for same target, it takes the same time. Since p2p network works by passing messages to peers for searching, the medium becomes a bottleneck for the overall performance of the network. If searching an item requires O (log n) messages and the messages are delayed due to the medium, the p2p network would slow down and eventually get congested.

Skip Graph has some space in the nodes at the topmost level of the pointers which can be exploited to further reduce the search time of sequential searches.

In our thesis, the proposed modification of Skip Graph viz. P-Skip Graph, aims to reduce the search time by probabilitiscally enumerating the target searched frequently and thus caching the link to the target in topmost level of the Skip Graph node and thereby reducing future search time of same target.

Chapter 3

Problem Statement

3.1 Problem Definition

P2P networks are distributed systems that became popular in the Internet applications in a short period of time due to efficient location of resources without any central authority. Survey reveals a list of desirable features for a peer-to-peer network such as decentralization, scalability, fault-tolerance, self-stabilization, data-availability, load-balancing, dynamic addition and deletion of nodes, efficient and complex query searching [14,15].

Various data structures have been proposed in literature for addressing these issues. There have been considerable work on Adjacency Matrix [5] in graph theory but it requires $O(n^2)$ space complexity and hence is not feasible for distributed network such as peer-to-peer. Skip-Lists [13] on the other hand are have also been researched but they also do not provide the true distributed structure since all the queries begin from starting of the top most levelled list thus providing a single point of failure. Some of the other data structures such as skip-webs or skip-nets have too high message congestion and others like Distributed Hash Tables [12] destroy the key-ordering, required to sustain complex operations such as range queries, and hence do not emerge as the best data structures.

Skip-Graphs [14] are the data structures that address almost all of the issues of distributed peer-to-peer network viz. fault tolerance, small sized nodes, fast queries and ordered data are addressed by skip graph. Researchers strive to find research gaps in the field of skip-graph and time and now propose a new data structure with improvement from the previous one.

This thesis also aims to optimize skip-graph in one such area. While surveying the literature it was observed that though there was a lot of work done on fault tolerance, authentication *etc*., a key area of optimizing the search queries for skip-graph has not been considered actively. The critical research gap that has been identified is that if one performs the same search query from the same node for the same target, the search has to be carried out from the scratch to identify the target node, thereby resulting in unnecessary duplicate messages in the network while search complexity still being the same O(log n).

The main objective of the thesis is to make the skip-graph network learn and adapt to the conditions of the network by creating direct link to the target based on the probability or

frequency of search for that target, and thereby reducing the further similar query's time complexity to O(1) without congesting the network.

3.2 Methodology

The objective is to take advantage of the top most level of the skip-graph node to use it point to the target that is searched frequently and is likely to be searched in the near future.
The methodology that is proposed to be followed is:

- Modify the current skip-graph node structure to accommodate a new level, and some vectors for storing count, and probability.
- Keep track of the count of searched items every time a search message comes with the search key field in it.
- Calculate the probability of the searched nodes and update them. If the probability rises above a certain limit (some threshold defined by a node), send a message to the target node for establishing a direct link to the target, to avoid any further delay in successive queries of the same node.
- Running similar queries for traditional skip-graph and p-skip graph network and analyzing the number of hops required to search the target node and hence calculate the time complexity for both.

Chapter 4

Design of P-Skip Graph

4.1 Peer-to-Peer Network

Peer-to-Peer network is a form of distributed network system where the nodes may be distributed across the globe and may be connected to the peer network over the internet. To store the data in graphs a fast and memory efficient data structure is required.

Data structures like adjacency matrix and list have been frequently used to store the graphical data of small size but as network grows the size of these data structure grows order of n^2 times. In such situations where memory consumption is huge and node size is dynamic, a data structure is required which can handle big data efficiently. From the detailed survey of the literature, Skip Graphs have evolved to be one of the best candidates for p2p network.

4.2 Traditional Skip Graph data structure

Skip graphs are mostly used in searching peer-to-peer networks as they provide the ability to query by key ordering (which tools based on DHT cannot), can tolerate large fraction of node fails and have efficient insertion , updation and deletion properties.

4.2.1 Skip Graph notations and terminology

The main components of a skip graph node are as follows:

Key : This is the value that distinguishes every node from the other node in the p2p network. The key is also required for ordering the nodes in the network. This is the main component of the skip node. Key can be any integer value that can be used for comparison such as < , > and =.

Data: Data can be any kind of data that is stored in the skip node. Typically for the sake of demonstration, an integer value is used for data in the node, but practically this can be anything specific to the need and nature of the network.

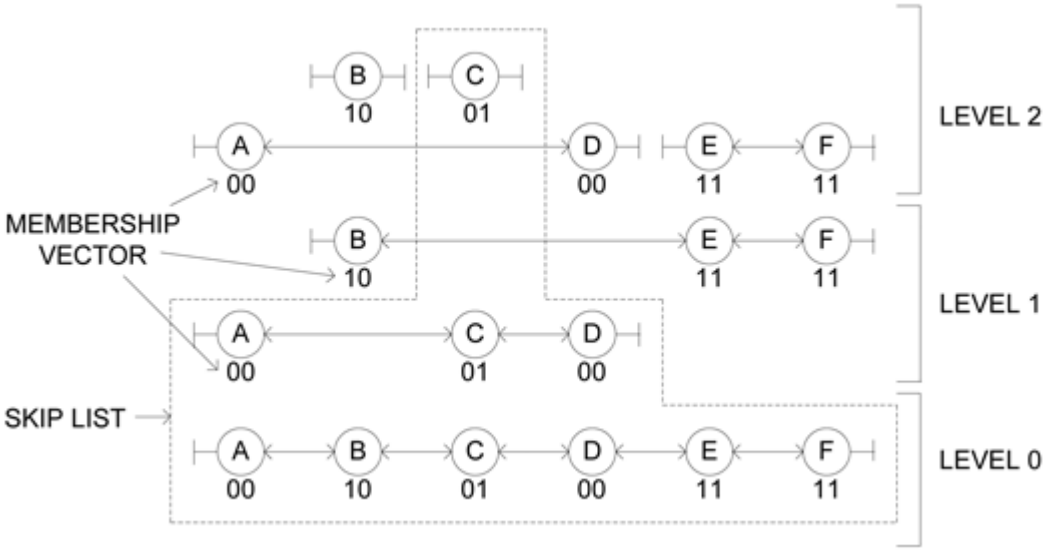


Figure 4.1: A skipgraph with levels = ceil(log n) = 3 [14]

Left and Right: Left and Right are generally a array of pointers to the neighbours of the skip node. Left array contains the list of pointers to the nodes which have the key value less than the current node's key value. Right array on the contrary has the pointers to the nodes with key values greater than the current node. Left and Right array are often indexed by Level

MaxLevel and Level: Maxlevel indicates the maximum number of levels the network has , or simply the max number of levels the node has. More specifically, it is the number of pointers stored in the left and right arrays.

Level (l) is often an integral value used to access the elements of the left and right arrays.

Specifically left[level] would mean that what is the left neighbour of the current node at level = level. Similarly right[level] is the right neighbour of the node at that level.

Membership Vector (m): Membership vector is an infinite sequence of bits used to decide the membership of the nodes at various levels.

Membership vector of a node x denoted by m(x) is an infinite random word over some fixed alphabet, although in practice only an O(log n) length prefix of m(x) needs to be generated on average. The idea of membership vector is that every doubly linked list in the skip graph is labelled by some finite word w, and an element x is in the list labelled by w if and only if w is the prefix of m(x) , or suffix of m(x) which depends on the implementation of the skipgraph. Figure 4.1 shows the nodes in the skip graph and their membership vectors.

The variables stored at each node are summarized in Figure 4.2. It is assumed that the pointer variables are maintained by an underlying deadlock-free implementation of a distributed sorted doubly-linked list, as in [14]. Further, it is assumed that operations on this doubly-linked list can be treated as atomic, and that it supports search, insert, and delete operations that each take O(k) time and O(k) messages starting from a node k hops away from its target. As per the assumption, operations on different doubly-linked lists can be performed in parallel without interference.

Figure 4.2: Variables stored in a typical node of skip graph.

4.2.2 Graph Storage using Skip Graph

Let G be a graph comprises of two main components (V, E) where V contains a set of vertices of graph and E contains a set of edges of graph. To store the whole graph corresponds to its mapping information consider a graph having few nodes as shown in Figure 4.3. The Vertices are the nodes in the p2p network and the links are the actual links between nodes of neighbourhood.

The data structure used to store this graph is skip graph. For storing each graph node a separate skip graph node is created and each link has a one to one correspondence between left and right pointers of the node at different levels. The links between the nodes are arranged in the left and right orders on the basis of the membership vector which is purely probablistic and random which just ensures that the nodes to not get clustered and hence the performance degrades. For best results, the membership vector should be non-repeating and random.

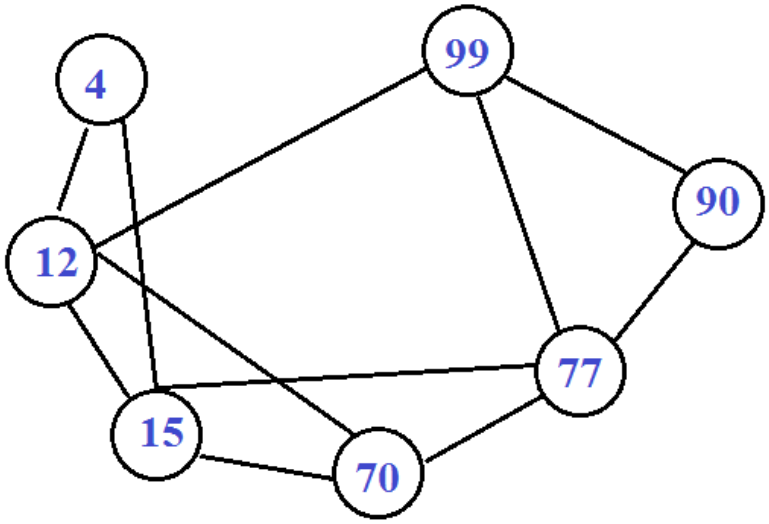


Figure 4.3: A Random p2p graph (Network)

Figure 4.4 depicts the Skip Graph version of the network in Figure 4.3.

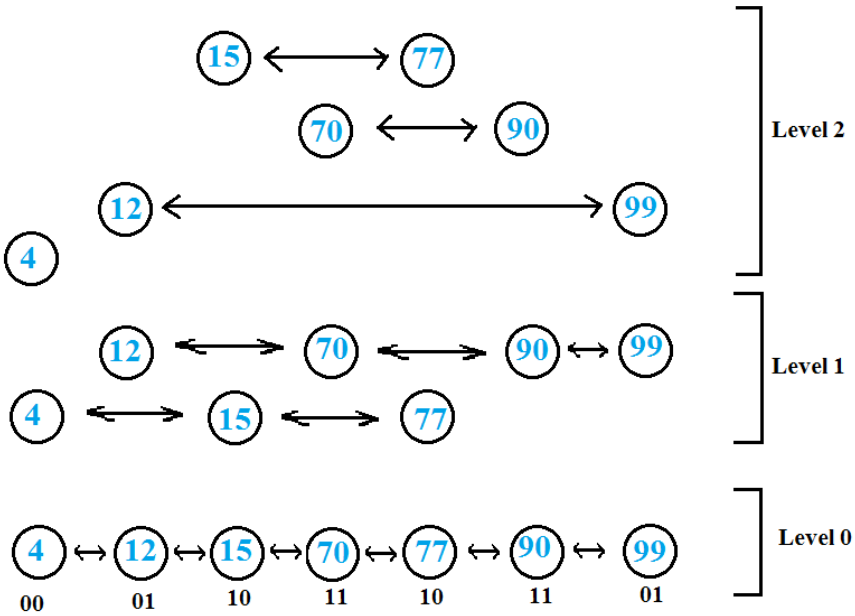


Figure 4.4 Skip graph for Figure 4.3 graph

4.3 Operations on a Skip Graph

Various operations in a Skip Graph can be performed using standard algorithms, such as insert new node, delete a node, traverse a node. For creation of Skip Graph, ‘n’ nodes would recall insert Algorithm:

4.3.1 Algorithm for insertion of a node with example

To insert a node in a Skip Graph:

Input:

- Key value of node as “key”
- Values of Left and Right neighbor for lateral updates.

Output:

- Insert the new node to its appropriate position in Skip Graph.

Insert a node In Skip Graph

A new node ‘u’ knows some introducing node ‘v’ in the network that will help it to join the network. Node ‘u’ inserts itself in one linked list at each level till it finds itself in a singleton list at the topmost level. The insert operation consists of two stages:

- Node ‘u’ starts a search for itself from ‘v’ to find its neighbours at level 0, and links to them.
- Node ‘u’ finds the closest nodes ‘s’ and ‘y’ at each level L >= 0, s < u < y, such that m(u) |(L+ 1) = m(s) | (L + 1) = m(y) |(L + 1), if they exist, and links to them at level L + 1.

Detailed pseudo code for the insert operation is given in Algorithm 1.

Algorithm 2: `insert` for new node n'

```
if introducer = n' then
    n'L_0 = ⊥
    n'R_0 = ⊥
else
    if introducer.key < n'.key then side = R
    else side = L
    send ⟨searchOp, n', n'.key, 0⟩ to introducer
    upon receiving ⟨searchOp, neighbor⟩:
    send ⟨linkOp, n', side, 0⟩ to neighbor
    level← 1
    while true do
        if n'L_{level−1} ≠ ⊥ then
            send ⟨buddyOp, n', level, m(n')_level⟩ to
            n'L_{level−1}
            upon receiving ⟨buddyOp, newBuddy, level⟩:
            if newBuddy ≠ ⊥ then
                send ⟨linkOp, n', R, level⟩ to newBuddy
        else if (n'R_{level−1} ≠ ⊥) ∧ (newBuddy = ⊥)
        then
            send ⟨buddyOp, n', level, m(n')_level⟩ to
            n'R_{level−1}
            upon receiving ⟨buddyOp, newBuddy, level⟩:
            if newBuddy ≠ ⊥ then
                send ⟨linkOp, n', L, level⟩ to newBuddy
            else break
        else break
        level←level+1
        n'L_level = ⊥
        n'R_level = ⊥
```

Figure 4.5: Insertion algorithm [14]

Example of Insertion operation in Skip Graph

Let us assume that a new node ‘20’ is to be added to the skip graph (Considering the network from figure 4.3 and 4.4)

Steps followed for insertion of Node ‘20’ are:

First find the exact location for insertion of new node by traversing the graph or searching for same node so that one gets address of the node storing the largest key less than the search key.

Then all the links and membership vectors are updated as per the new node’s requirements (Figure 4.8).

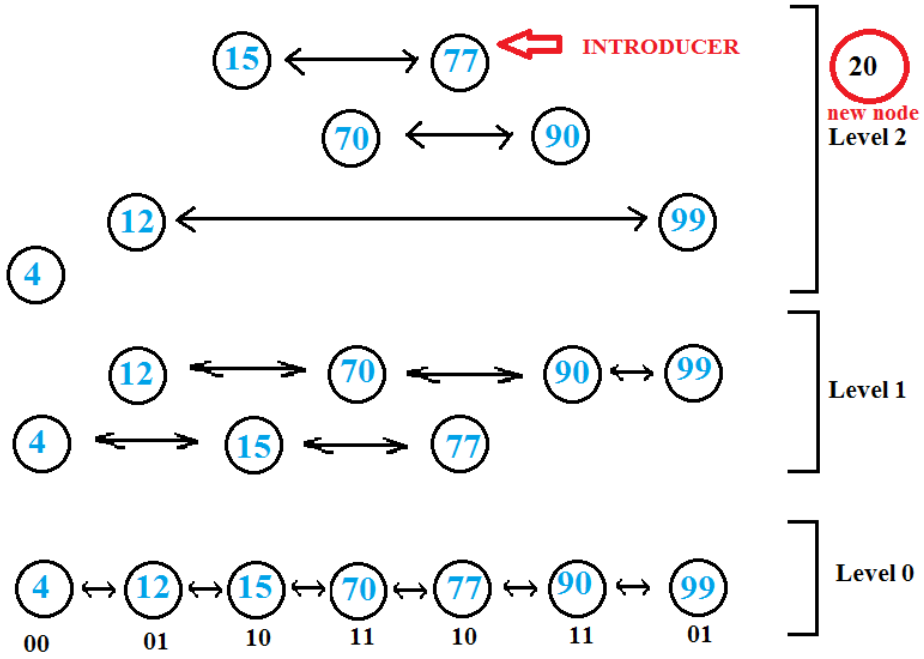


Figure 4.6: Skip Graph for Insertion

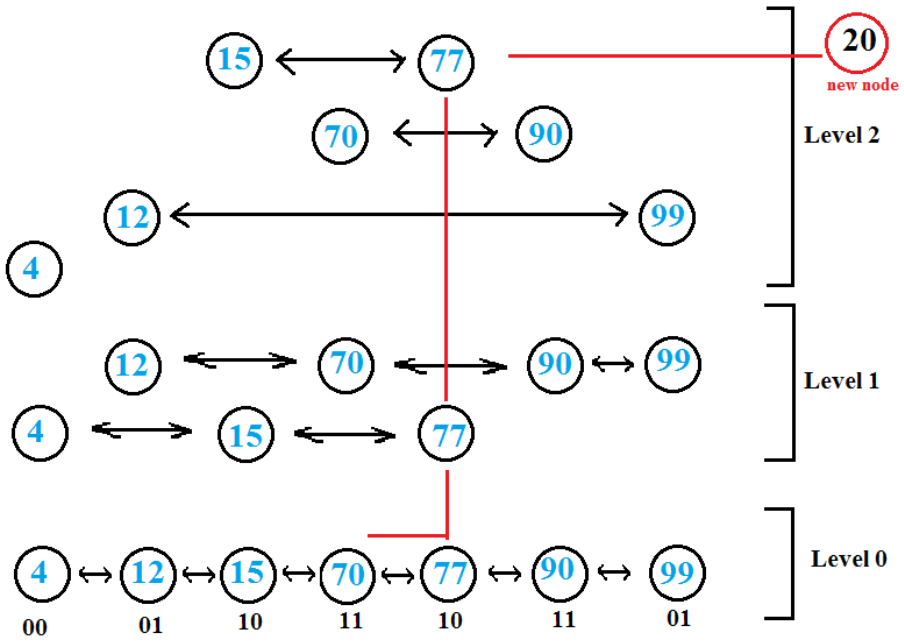


Figure 4.7: Insertion of new node ‘20’

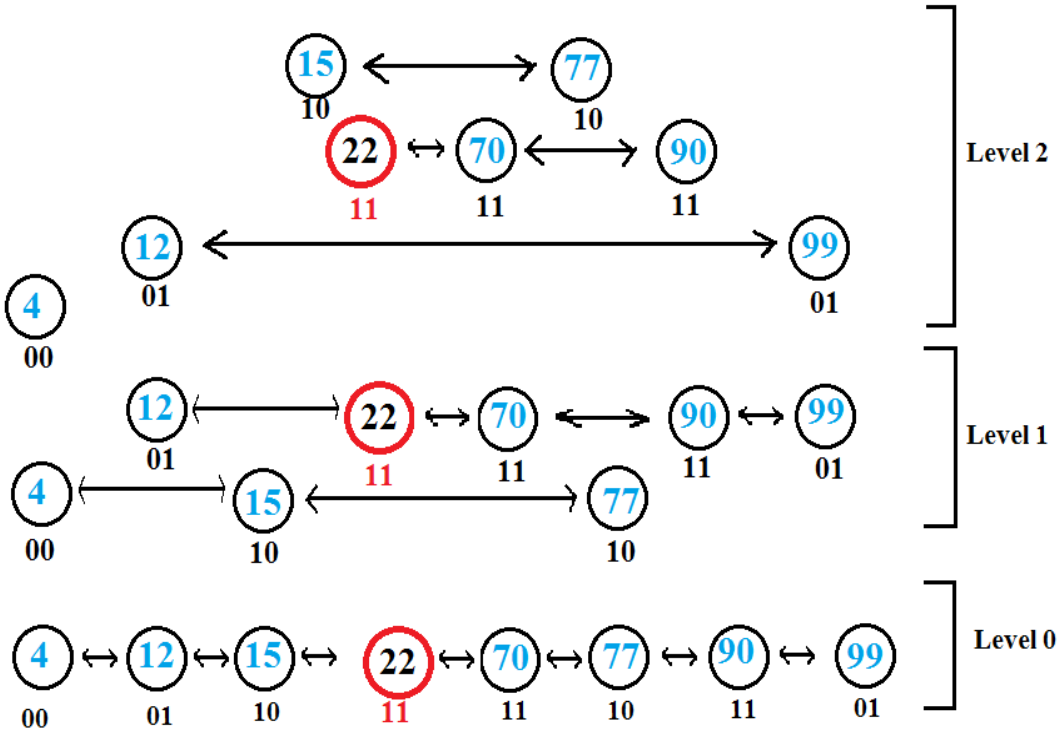


Figure 4.8: Graph after Inserting node ‘20’

4.3.2 Algorithm for searching of a node with example

Input:

- Key value of node to be searched as “searchkey”
- Key value of initiating node as "startnode"
- A code to tell the nodes which operation to execute.

Output:

- If it exists the address of the node storing the search key is returned, But if node does not exists then address of the node storing the largest key less than the search key is returned.

Search Algorithm for a node in Skip Graph

- The search starts at the topmost level of the node seeking a key and it proceeds along each level without overshooting the key, continuing at a lower level if required, until it reaches level 0.

Algorithm 2: search for node v

```
1  upon receiving ⟨searchOp, startNode, searchKey, level⟩:
   // If the current key is the search key, return
2  if (v.key = searchKey) then
3  |  send ⟨foundOp, v⟩ to startNode
   // Coming from left, current key is less than search key
4  if (v.key < searchKey) then
   |  // Go one level down till a right neighbor with key smaller than the search key is reached
5  |  while level ≥ 0 do
6  |  |  if ((v.neighbor[R][level].key ≤ searchKey) then
7  |  |  |  send ⟨searchOp, startNode, searchKey, level⟩ to v.neighbor[R][level]
8  |  |  |  break
9  |  |  else level←level-1
   // Coming from right, current key is greater than search key
10 else
11 |  while level ≥ 0 do
   |  |  // Go one level down till a left neighbor with key larger than the search key is reached
12 |  |  if ((v.neighbor[L][level]).key ≥ searchKey) then
13 |  |  |  send ⟨searchOp, startNode, searchKey, level⟩ to v.neighbor[L][level]
14 |  |  |  break
15 |  |  else level←level-1
   // Reached the bottom-most level, key is not found
16 if (level < 0) then
17 |  send ⟨notFoundOp, v⟩ to startNode
```

Figure 4.9: Searching algorithm [14]

Example of Search operation in Skip Graph

To search Node '90' (considering the Figure 4.3 and Figure 4.4) following steps should be followed:

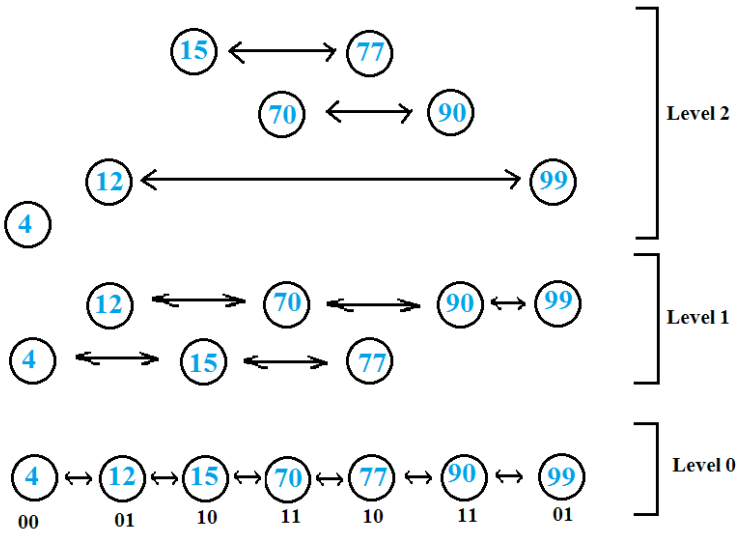


Figure 4.10: Skip Graph for Search algorithm '90'

- Node 4 starts the query therefore startnode = 4
- 4 sends a message to its right to continue search for the key (if the current level the neighbour exists,). If the neighbour does not exist at that level, the level is decremented and the same procedure is followed.
- When the key is found, the node sends the message to the startnode.

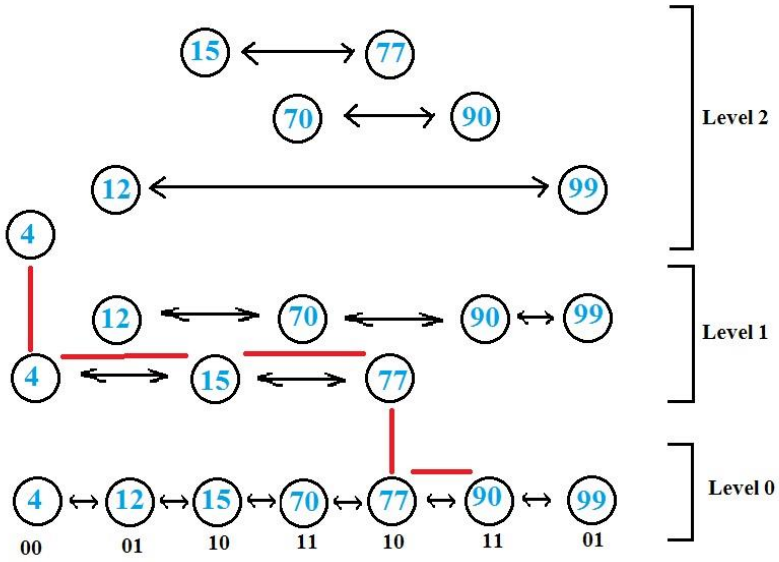


Figure 4.11: Search operation for node ‘90’

4.3.3 Algorithm for deletion of a node with example

To delete a node in a Skip Graph:

Input:

- Key value of node as “key”

Output:

- Node removed from the Skip graph and all links updated

Deletion of Node From Skip Graph

When node ‘20’ wants to leave the network, it deletes itself in parallel from all lists above level 0 and then deletes itself from level 0.

Algorithm 3: delete for existing node u

```
1 for ℓ ← 1 to u.maxLevel in parallel do
2 |   delete u from list at level ℓ
3 delete u from list at level 0
```

Figure 4.12: Deletion algorithm [14]

Example of deletion of node from Skip Graph

First the deletion node should be identified on all level Suppose Node '20' needs to be deleted from the graph .

- Start deletion process from Level max to 1. Delete selected node from all these levels.
- The node sends the update messages to the neighbouring nodes to update their neighbours, by sending the detail of the right neighbour at level L to the left neighbour at the same level.
- After sending the message, the node decrements the level and proceeds in the same manner until the node also deletes itself from level 0.
- When the neighbours update their links the node deletes itself. (i.e the host system may shutdown safely now)

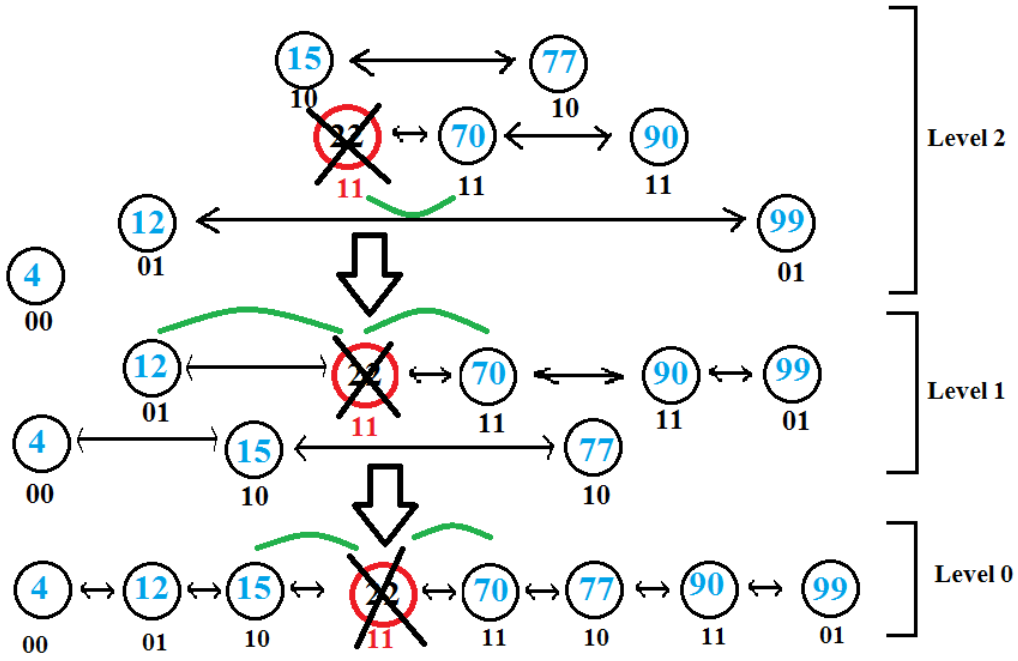


Figure 4.13: Deletion operation in skip Graph

Skip graph obtained after deleting Node ‘20’ is (Figure 4.14):

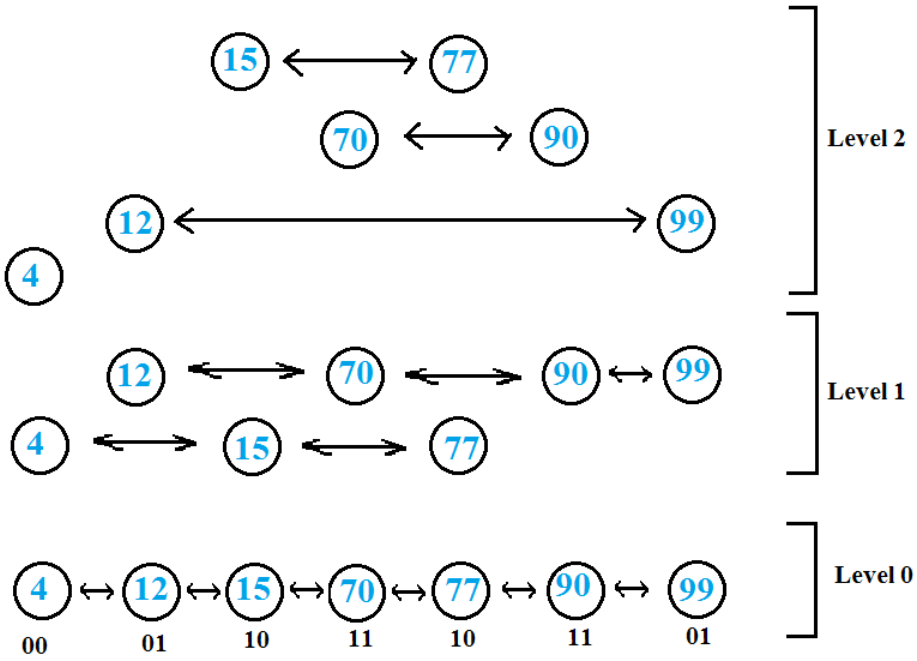


Figure 4.14: Graph after deleting node '20'

4.4 Proposed Approach: P-Skip Graphs

A novel data structure which called as P-Skip Graph is proposed which is derived from traditional skip graph. In this data structure a new level has been added to the top of the skip graph where initially all the nodes are in themselves a linked list with their right and left neighbours being empty. This level is sometimes present in the traditional skip graph when all the nodes have entirely random but different membership vector.

Moreover traditionally the number of bits used in the membership vector are ceil (log n) -1. For e.*g.* if there are 8 nodes , levels = 3 , viz. 0,1,2, where level 0 has all the nodes connected in doubly linked list , level 1 has 2 linked lists whose least significant bit match. Level 2 has 4 lists where all the nodes having least 2 significant bits same.

However authors in [14] and [15] also highlight the fact that there can be another level *i.e.* level 3 but it would have all the nodes in their own linked lists, which though would increase the number of levels to log n + 1 but the overall search complexity still grows at O(log n + 1) which is again O(log n). But since the topmost level would contain no links to the neighbors', it is often ignored.

4.4.1 P-Skip Graph and the Next Level

It is assumed that pnode would refer to the node of a p-skip graph.

A pnode consists of the following new fields along with the traditional fields:

Count_Vector: This vector keeps track of the number of times a particular key is searched by starting from this node. Every time a key is searched, the node updates its count_vector to reflect the number of times a particular key has been searched via this node.

Probability_Vector: Probability vector keeps track of the probability that a particular node will be searched in the near future by taking into account the previous searches through that node.

Top-most P-Level: The topmost level, which is often left out in traditional skip graphs is now included as P-Level in the pnode. This P level is put into use by means of probability_vector. Since probability vector has all the calculated probabilities stored in it with respect to the keys that are searched, the P-level is used to create a link to the node with the highest probability.

Complexity of Skip graph

When a key is searched in a traditional skip-graph (especially when the nodes are distributed across the internet) with high number of nodes, the search takes place with O(log n) messages being passed between peers (which may be routed through the internet via standard routing protocols).

Lets assume that there are approx 65535 members in a p2p network and they share a skip-graph network for communication. The max number of levels that would be possible would be log (65535) = 16. When a key is searched by a node, there are O(log n) messages passed to get the key *i.e.* 16 messages worst case.

Now consider a scenario that a particular resource has become very popular and is requested time and again from a particular node.

Lets assume that a node receives 100 requests for the same resource.

Every time the node receives a request, it sends the message via the same nodes, thereby congesting the network with 16 * 100 = 1600 messages.

This is where P-Level is used

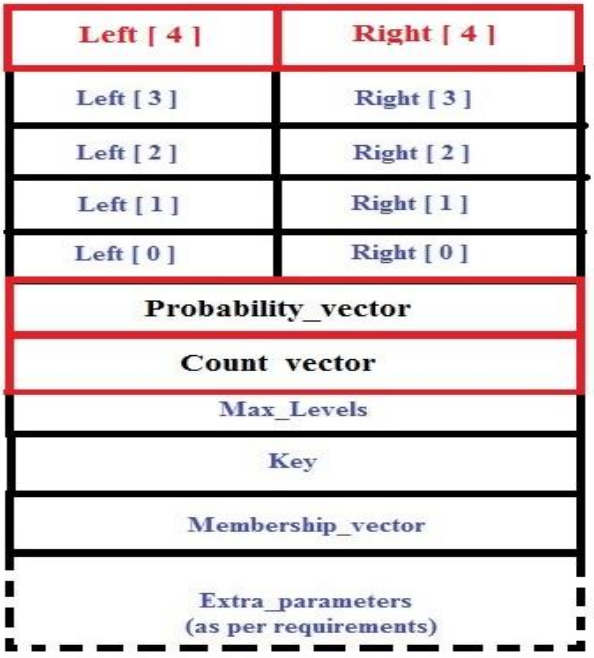


Figure 4.15: Variables stored in a typical node of p-skip graph

As the count of a particular key increases, the probability of the key being queried in future also increases.

Every node can set up its own threshold value. When the probability of a particular node exceeds the threshold value, the p-level of the node creates a link with the target key so as to directly address the target node and route the message via the internet rather than routing the message to the nodes of the skip graph and creating unnecessary chaos. Thus once the target node is included in the P-level of the node all subsequent queries can be done in O(1) time complexity while giving the benefit to the network nodes to continue with their own operations thereby making the network congestion free and improving the effective search time.

4.4.2 P-Skip Graph operations

The operations of the P-skip-graph are exactly the same as the traditional skip graph. There is no change in the existing algorithms. However, only the search algorithm needs to be modified for constructing the P-level.

Since p-level is just another level in a node, only the maxlevel needs to be incremented by 1.

4.4.2.1 Modified search algorithm for p-skip-graph

The algorithm is exactly the same as traditional skip graph , with the only difference being that when the node receives the request to search a node key it performs the following steps :

- It checks for the searchKey not being the same node *i.e.* it checks that the node to be queried is not itself.
- If the node is someone different it increments the count vector to reflect that a new query has been generated.
- It then updates the probability vector to reflect the possibility of any future query.
- If the probability of the node exceeds the threshold value, the node calls the getaddr algorithm to get the address of the key to be searched.

```
Algorithm : search for node n
upon receiving ⟨searchOp, startNode, searchKey,
level⟩:
if n.key = searchKey then
   send ⟨searchOp, n⟩ to startNode
 if n.key != searchKey then
   increment count [searchKey]
   for all i in prob:
        update prob [ i ]
   if prob[ searchKey ] >= threshhold then
        getaddr< getOp , source , dest >
if n.key < searchKey then
   while level ≥ 0 do
      if (nR_level).key < searchKey then
         send ⟨searchOp, startNode, searchKey,
         level⟩ to nR_level
         break
      else level←level-1
else
   while level ≥ 0 do
      if (nL_level).key > searchKey then
         send ⟨searchOp, startNode, searchKey,
         level⟩ to nL_level
         break
      else level←level-1
if level < 0 then
   send ⟨searchOp, n⟩ to startNode
```

Figure 4.16: The modified search algorithm for the p-Skip Graph

4.4.2.2 Getaddr algorithm for P-Skip Graph

When a node receives getOpt option, it performs the following operations :

- It checks if the destination is the same as the node. If it is so , it sends a reply to the source giving it the address of the node so that the source could direclty address the message to the target without following the traditional approach.
- If the destination is not the node itself , it checks the key of the destination and forwards it according to the normal search algorithm (level by level)

Algorithm : getaddr < getOpt , source , dest >

```
upon receiving <getOpt , source , dest >
 if n.key == dest.key then
    send < replyOpt , source, dest , dest.addr > to source
else
     if n.key < dest.key then
         while level > 0
              if nR_level  != NULL then
                   send < getOpt ,source , dest > to  nR_level
                   break
              else level <- level - 1
   else
         while level > 0
              if nL_level  != NULL then
                   send < getOpt ,source , dest > to  nL_level
                   break
             else level <- level - 1

if level < 0 then
    send < errorOp , source  , dest > to source
```

Figure 4.17: GetAddr algorithm for the p-Skip Graph

4.4.2.3 Reply algorithm for the p-skipgraph

When a node receives replyOpt option, it checks the target key and then adds the address of the target to its left or right p- level according to the order of traditional skip graph.

Algorithm : reply for the address

upon receiving < replyOpt , source , dest , dest.addr >
if *dest.key* < *n.key* **then**
nL_p = dest.addr
else
nR_p = dest.addr

Figure 4.18: Reply algorithm for the p-Skip Graph

Chapter 5

Implementation Details

5.1 Tools used

The implementation of the p-skip graph peer-to-peer network has been done entirely in linux environment using python (because of its rich socket library) and bash (for automating the process of file creation using bash scripts and utilities like sed, cut and bc. Therefore enumeration can be done as:

- Platform used : Linux (Backtrack 5R3)
- Language used : Python (for programming) and bash (for scripting)
- Other Utilities used : Netcat (for sending and receiving udp packets)

5.2 Implementation details

Netcat is a very useful tool used for network analysis. The nodes of the peer-to-peer network are created through files and script. However to get to know what messages are being transmitted between the nodes netcat has been used to send and receive udp packets while still using the nodes.

Netcat gives the ability to the user to emulate a node by sending crafted packets and then capture the reply while still using the underground network architecture designed. All the nodes of the peer-to-peer network are executed on the same machine with different port numbers.

For the sake of simplicity the key and the data has been assumed to be equal to the port number on which the node is running. While still maintaining the characteristics of the peer-to-peer network this assumption greatly simplifies the execution process.

Figure 5.1 shows the "node.py" file that contains all the logic and the algorithms for p-skipgraphs. This file shows some of the variables used in the node such as key, left [], right [] lists, count [] and prob [] vectors as well.

To implement the p-skipgraph , the membership vector (just as the traditional skip graph) is required for the nodes. In this experiment the script file generates totally random membership vector without repeting. This is purposely done so that nodes do not get clustered in all the levels thereby reducing the benefit offered by skip graphs to the peer-to-peer network

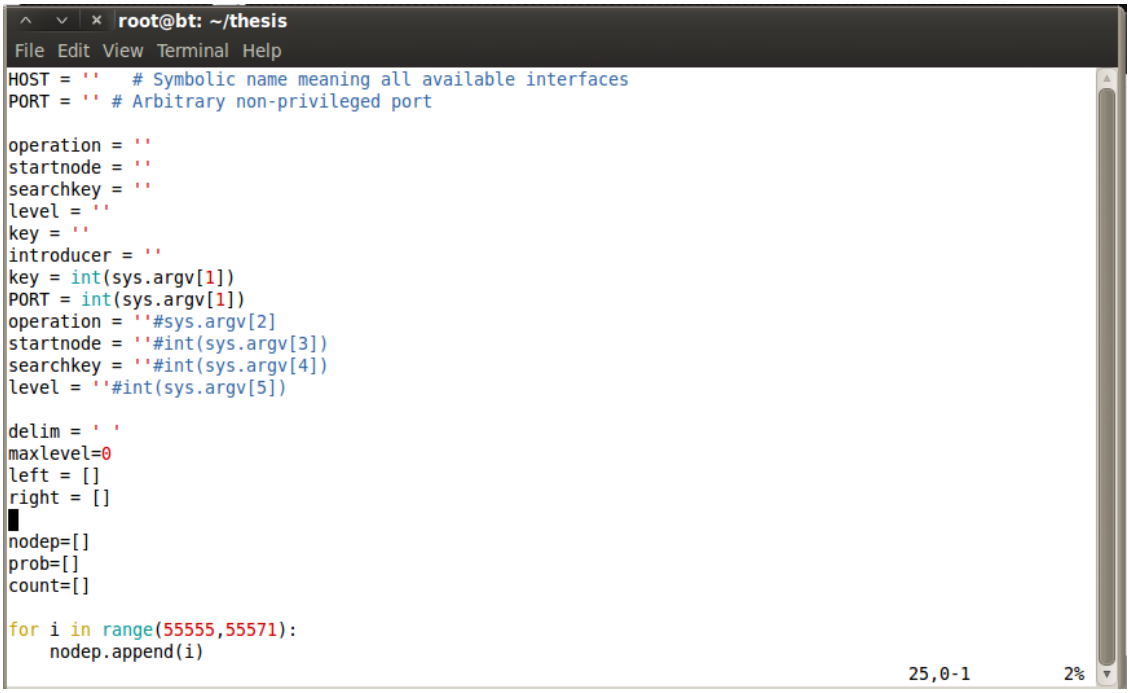


Figure 5.1: "node.py" file having variables of a p-skipgraph node

Figure 5.2 shows the "membershipvector.py" script file that generates random membership vectors and puts them in the "membership.txt" file. (Figure 5.3)

Figure 5.2: "membership_vector.py"

Figure 5.3: "membership.txt" file

After the membership vectors are generated links.py script is used. This script file generates links for the nodes of the p-skipgraphs. This script file compares the membership vectors of all the nodes and creates the left and right links of the nodes for various levels. These links are stored as pairs of left and right link in the text file. Figure 5.4 shows the list0.txt,list1.txt, list2.txt, list3.txt files that have been generated by links.py script.

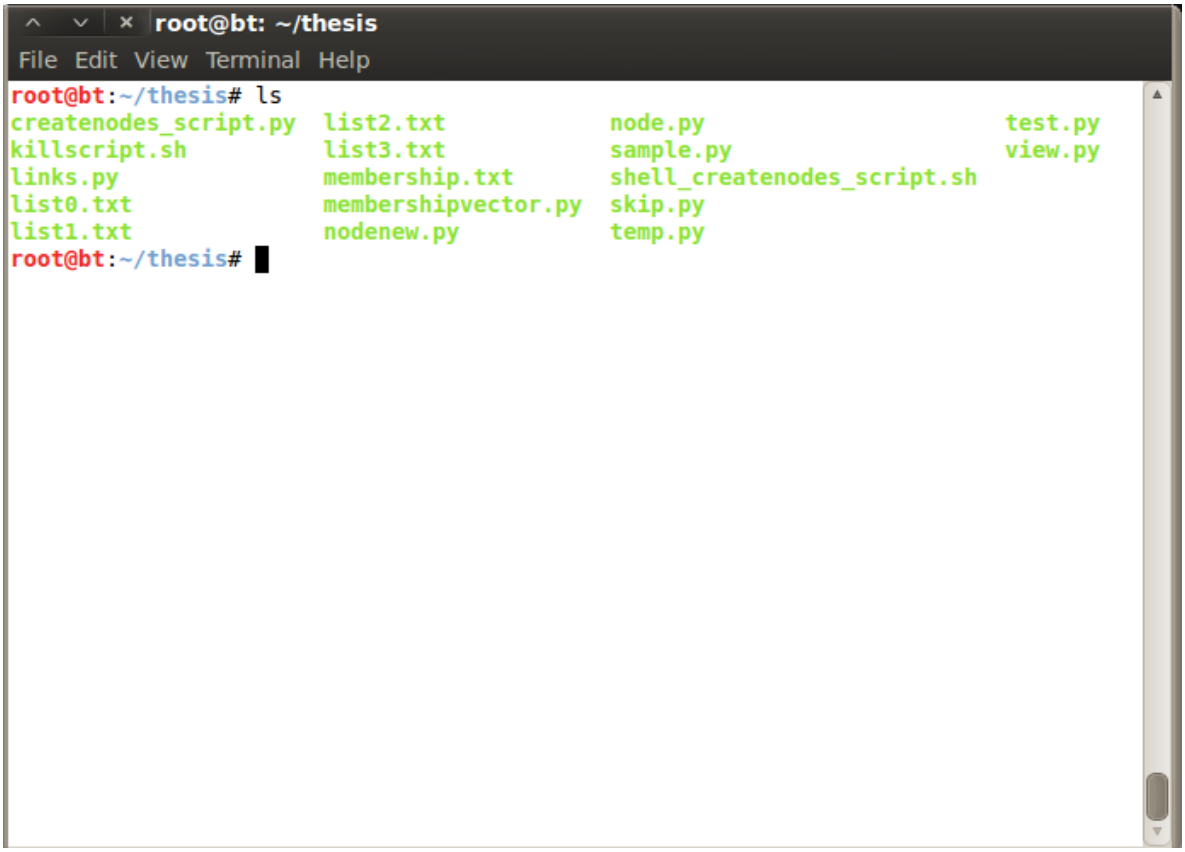


Figure 5.4: "list0.txt" , "list1.txt" , "list2.txt" and "list3.txt" files.

After the list files are ready with all the links of the nodes at various levels stored into it, some scripts are generated to create number of nodes of the p-skipgraph. This task is achieved by two scripts. The first script "createnodes_script.py" generates all the necessary details required for each node seperately and gives the output to the next script.

The next script "shell_createnodes_script.sh" is actually a shell script that gathers all the necessary information generated by the previous script and calls the main python program "node.py" with all the necessary commandline arguments to be passed to the node. This script generates the p-skipgraph nodes and send them in the background treating them as daemons.

Figure 5.5 shows the snapshot of "createnode_script.py".

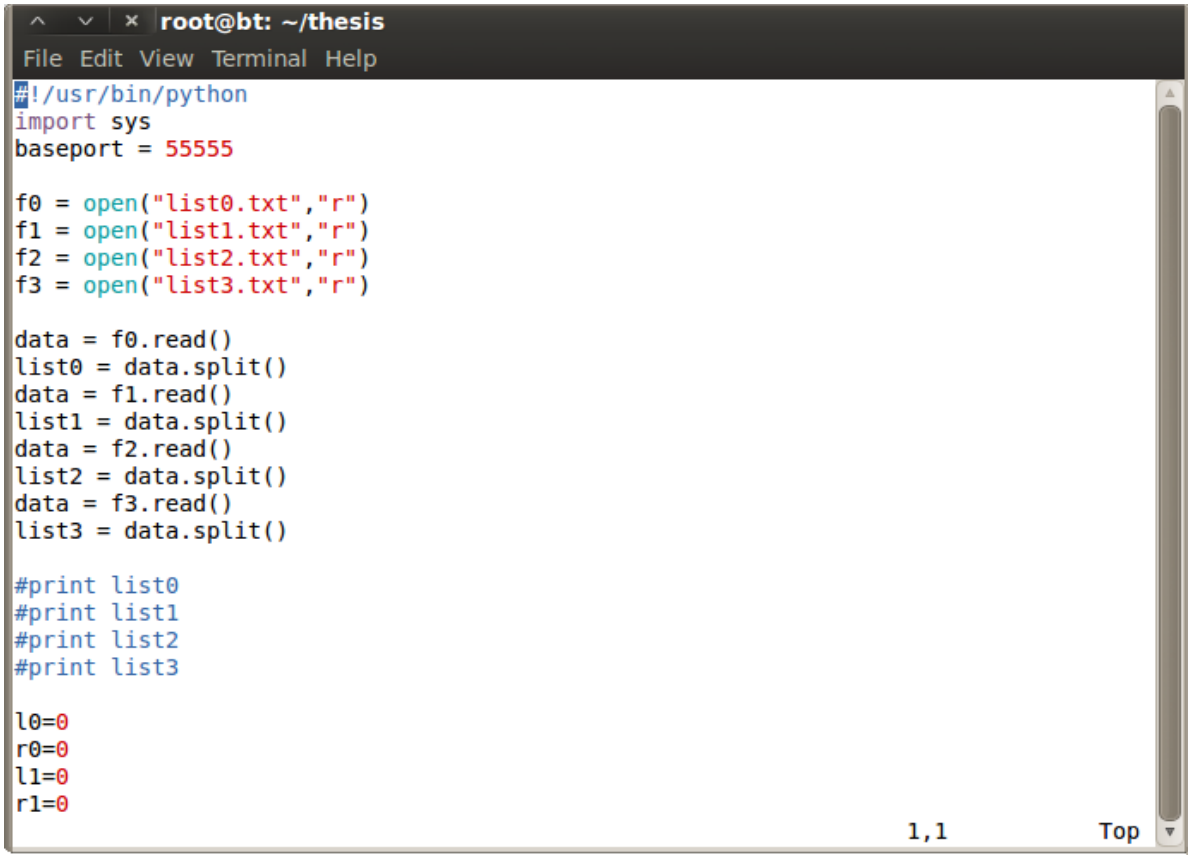


Figure 5.5: "createnodes_script.py"

This shell script creates nodes in the background and makes the peer-to-peer network up and running.

Each node runs on a dedicated port on the same machine i.e localhost and requires to know the port number of the other node as the address for communication with the nodes in the neighbourhood. Each node that is on the edge of the graph , stores -1 as the port numbers to depict NULL values of the links that donot correspond to any legitimate node.

The "shell_createnodes_script.sh" script passes the links values of the left and right arrays through commandline arguments. These arguments are then stored independantly by each node in their respective left and right lists of pointers.

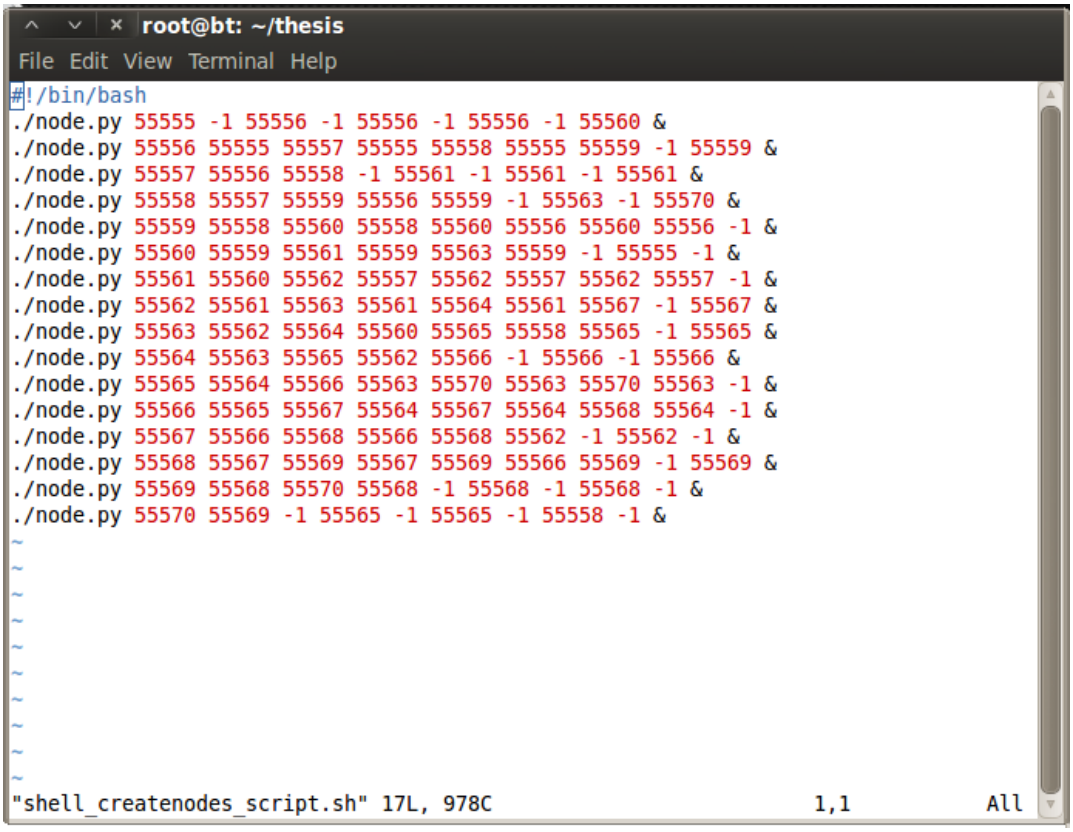


Figure 5.6: "shell_createnodes_script.sh".

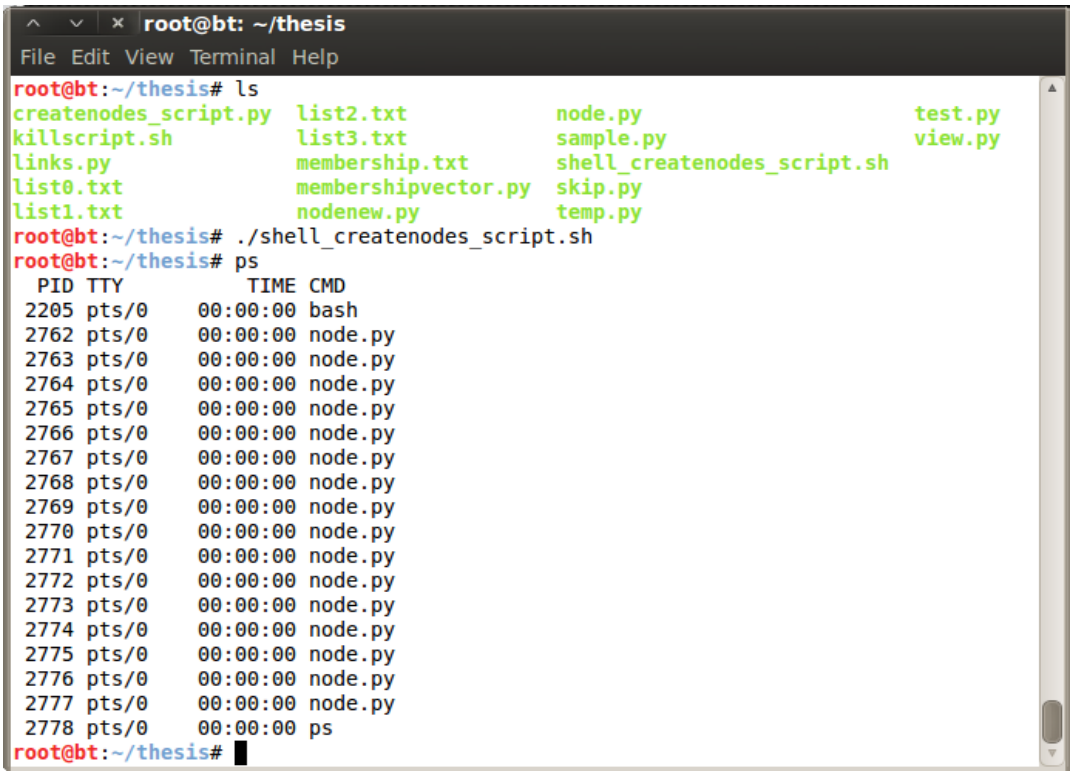


Figure 5.7: Nodes created and running in the background.

Figure 5.8 shows the search operation in the skipgraph node. The message has been crafted as "SEARCH startnode searchKey level". When the node with a particular key is found it replies back to the startnode with the reply message.

The reply consists of the message "FOUND startnode searchkey level key" where the last key is the key of the node that matches searchkey. It is merely added to ensure that legitimate node is replying to the search message and that there is no message that is bounced back to the startnode.

For netcat to lookup the messages , messages are send through netcat while it portrays to be a part of the network by giving the startnode port number to be the port number where netcat is running and sending the message to any node of the network.

Figure 5.8: Search operation code

Figure 5.9 shows the query operation code in python. This query operation is the typical landmark of skip graph since other data structures that realize peer-to-peer network like DHT cannot efficiently execute a range query on the network nodes due to the fact that key ordering is destroyed when the keys are hashed.

Skip Graph takes the key ordering to its benefit to support complex queries such range queries.

The Figure 5.10 shows the range query result. For the sake of demonstration the query has been assumed to be greater than or equal to query. It also shows the query that asks all the nodes greater than 55555 with startnode being the port number of netcat utility.

The query result message arrives with "*queryresult key l[0] r[0] l[1] r[1] l[2] r[2]* ..... " message , where queryresult is just the opcode , key is the key of the sending message , l[i] r[i] are the link descriptions that have been stored in the left and right list of the node

Figure 5.9: "query"operation code

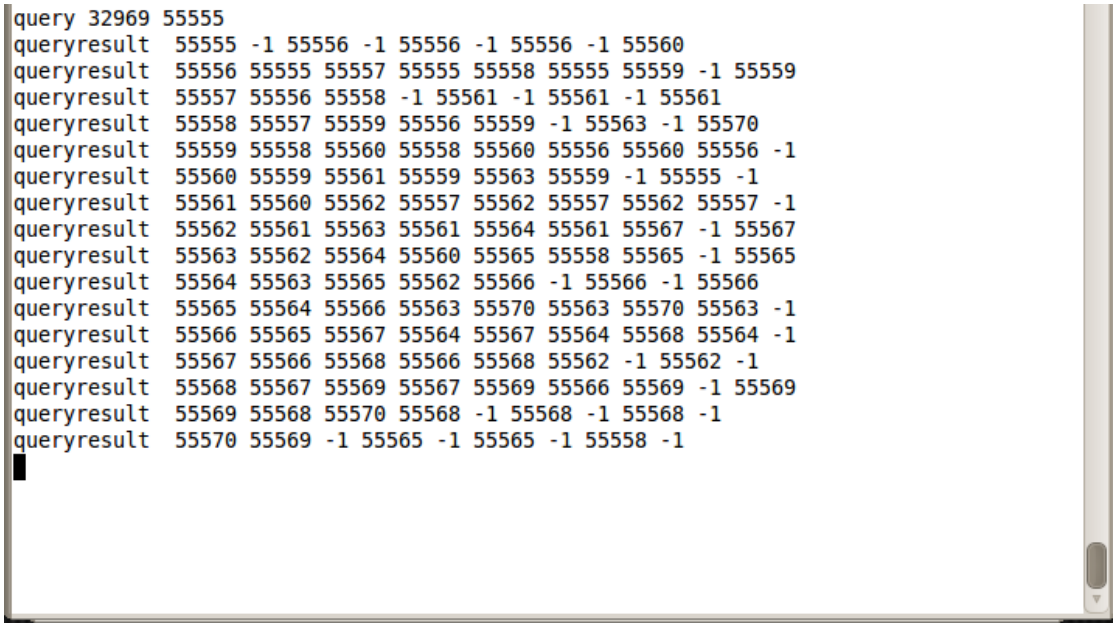


Figure 5.10: Netcat receiving queryresult from the nodes.

5.3 P-Skip Graph vs. Traditional Skip Graph

The modification of the skip graph i.e the p-skip graph in this thesis leverages the topmost level of the nodes left and right arrays and fills them probabilistically with appropriate link description. Figure 5.11 shows the result of the search operation performed on skip graph and figure 5.12 shows the result of the search operation performed on p-skip-graph.

This search result message contains an additional field to calculate the number of hops the message travelled to reach the target. Figure 5.11 depicts that even though the same query is being performed for the same target from the same source, there is always a search carried out from scratch. This bottleneck is removed by p-skip graph.

```
search 32969 55567
FOUND 55567 32969 55567 5
search 32969 55567
FOUND 55567 32969 55567 5
search 32969 55567
FOUND 55567 32969 55567 5
search 32969 55567
FOUND 55567 32969 55567 5
search 32969 55567
FOUND 55567 32969 55567 5
search 32969 55567
FOUND 55567 32969 55567 5
```

Figure 5.11: Search operation on traditional skip graph network.

```
search 32969 55567
FOUND 55567 32969 55567 5
search 32969 55567
FOUND 55567 32969 55567 5
search 32969 55567
FOUND 55567 32969 55567 5
search 32969 55567
FOUND 55567 32969 55567 1
search 32969 55567
FOUND 55567 32969 55567 1
search 32969 55567
FOUND 55567 32969 55567 1
```

Figure 5.12: Search operation on p-skip graph network.

As has already been explained in the algorithm in Section 4.4.2.1 that p-skip graph takes into account the probability that a particular node would be searched frequently and includes the link to that node in the *p-level* of the source node.

This makes a direct link to the target and hence reduces the further time complexity of the successive search queries for the same node. Figure 5.12 depicts the number of hops decreasing to 1 when the target gets added to the *p-level* of the source.

Chapter 6
Conclusion and Future Scope

6.1 Conclusion

Skip Graph, due to its properties, is best suited for peer-to-peer network than any other data structure. Though enormous work has been done to further improve and optimize the properties of skip graphs, the possibility of trend in the search queries in the peer-to-peer network based on skip graph was not attended.

P-Skip Graph addresses this issue in peer-to-peer network by utilizing an extra level of the skip graph. Since this extra level would also be included in normal search queries thereby making the effective number of levels to *log n +1* and the search complexity to be *O(log (n) + 1)* which essentially grows as fast as *O(log n)*. Hence there is no degradation of traditional search query. Moreover since the *p-level* , that is utilized in P-Skip Graph, will contain a link to the most frequently searched target, the search time from the node containing the p-level pointer would to the target would be *O(1).* Also since the count and the probability calculation is done every time a new search request is received, the time complexity to increment the count and the total count to find the probability is also of the order of *O(1).*

Hence it is concluded that P-Skip Graph improves the search complexity of the traditional skip-graph in peer-to-peer network while still displaying the features of trivial skip graph efficiently.

6.2 Future Scope

The experiment has been performed locally on the same host machine with nodes running as different process on different ports. Thus the factors like internet congestion, message jitter have not been considered. The experiment, however, can be performed in the real world scenario with just a few minor changes in the "node.py" file to adjust to the real time peer-to-peer network over the internet.

Due to the hardware limitations, the experiment has been performed with 16 peer-to-peer nodes. However the with the availability of different host computers for running single node per host, the experiment could be extended to any number of nodes. Moreover the probability calculation technique is very trivial and can be substituted with more sophisticated and reliable statistical testing technique.

Since a peer-to-peer network relies a lot on message passing between nodes, P-Skip Graph can also be combined with some of the known authentication techniques to validate the message and data being transferred between the nodes.

References

[1] Coulouris, George; Jean Dollimore, Tim Kindberg, Gordon Blair, "*Distributed Systems: Concepts and Design" , (5th Edition*). Boston: Addison-Wesley.2011

[2] Andrews, R.Gregory "*Foundations of Multithreaded, Parallel, and Distributed Programming",* Addison–Wesley, 2000.

[3] "Histroy of email"[online] available: "*http://www.nethistory.info/History/email.html"*,[Accessed 2 June.2013].

[4] Herlihy, Maurice P.; Shavit, Nir N. "*The Art of Multiprocessor Programming"*, Morgan Kaufmann, 2008.

[5] Linial, Nathan "*Locality in distributed graph algorithms*", SIAM Journal on Computing, page 193–201,1992

[6] Papadimitriou, H. Christos, "*Computational Complexity"*. Addison–Wesley, 1994

[7] Andrews, Gregory R. "*Foundations of Multithreaded, Parallel, and Distributed Programming"* , Addison–Wesley,2000.

[8] Rüdiger Schollmeier, "*A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications"*, Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE (2002)

[9] "Peer to Peer networks" [online] available: "*http://www-sop.inria.fr/mistral/personnel/Robin.Groenevelt/Publications/Peer-to-Peer_Introduction_Feb.ppt"* , [Accessed 20,May,2013]

[10] J. J. HOPFIELD, "*Neural networks and physical systems with emergent collective computational abilities*", Proc. NatL Acad. Sci. USA Vol. 79, pp. 2554-2558, April 1982 .

[11] Morgan Kaufmann , Plaszczak, Pawel; Rich Wellner, Jr , "*Grid Computing The Savvy Manager's Guide"* ,2006.

[12] "Peer to Peer routing "[online] Available: "*http://project-iris.net/talks/dht-toronto-03.ppt*", [Accessed 26,May,2013]

[13] U Wilder, G. Shah, "*Skip lists: a probabilistic alternative to balanced trees*", *Communications of the ACM* (6): 668–676.1990

[14] "Skip Graphs"[online] Available: "*http://www.cs.yale.edu/Skip Graphs*", by James Aspnes; Gauri Shah, Department of Computer Science, Yale University , 2006 [Accessed 28 March 2013]

[15] M.T.Goodrich M.J.Nelson,J.Z.Sun , "*The Rainbow Skip Graph: A Fault-Tolerant Constant-Degree Distributed Data Structure*", January 22,26, SIAM, May 2009.

[16] Chapman & Hall/CRC , Ghosh, "*Distributed Systems – An Algorithmic Approach"*, Sukumar 2007,

[17] Keidar, Idit, "*Distributed computing column 32 – The year in review*", *ACM SIGACT News,2008*

[18] Naor, Moni; Stockmeyer, Larry , "*What can be computed locally?"*, *SIAM Journal on Computing,1995*

[19] Attiya, Hagit and Welch, Jennifer , "*Distributed Computing: Fundamentals, Simulations, and Advanced Topics "*, Wiley-Interscience, 2004.

[20] Garg, Vijay K , "*Elements of Distributed Computing"*, Wiley-IEEE, 2002.

[21] Sylvia Ratnasamy , Paul Francis , Mark Handley , Richard Carp , Scott Shenker "*A Scalable content addressable network*", In Proceedings of the ACM SIGCOMM , pages 161-170 , 2001

[22] Ion Stoica , Robert Morris , David Karger, Frans Kaashoek, Hari Balakrishna , "*Chord: A scalable peer-to-peer lookup service for internet applications*", In Proceedings of the SIGCOMM 2001, pages 149-160 , 2001

[23] Antony Rowstron,Peter Druchel , "*Pastry: Scalable , Distributed object location and routing for large scale peer-to-peer systems*.", In Proceedings of 18th IFIP /ACM International Conference of Distributed Systems Platform (Middleware 2001) ,Heidelberg , November, 2001

[24] D. Joseph , John Kubiatowikz ,Satish Rao ,and Ben Y zhao , "*Tapestry: An infrastructure for fault tolerant wide area location and routing*." , Anthony Technical Report UCB/CSD-01-1141 , University of California , Berkeley, April 2001

[25] Dahlia Malkhi, Moni Naorand David Ratajczk , "*Viceroy: A Scalable and dynamic emulation of the butterfly*.", In 21st ACM Symposium on Principles of Distributed Computing , pages 183-192 , Monterey , CA, USA , July 2002

[26] "NAPSTER tool in Peer to Peer" [online] Availbale: http://www.napster.com [Accessed 27 May 2013]

[27] "Peer to Peer tool GNUTELLA" [online] Available: http://gnutella.wego.com [Accessed 5 June 2013]

[28] "Peer to Peer tool FREENET" [online] Available: http://www.freenet.sourceforge.net [Accessed 7 June 2013].

[29] P .Kirchenhoefer , C.Martinez , and H.Prodinger , "*Analysis of an optimized search algorithm for skiplist*", Theoretical Computer Science , page 119-220 , 26 June 1995

[30] Shabeera T.P , Priya Chandran , Madhu Kumar S D ,"*Authenticated and Persistent Skip-Graph : An Data Structure for Cloud Based Data-Centric Applications* " , ICACCI '12 , ACM , August 03-05 2012

List of Publications

Amrinderpreet Singh, Dr. Shalini Batra, "*P-Skip Graph : An Efficient Data Structure for Peer-to-Peer Network*", in the INTERNATIONAL JOURNAL OF MANAGEMENT & INFORMATION TECHNOLOGY, July Edition, (Communicated).