Methods for Efficient and Secure Service Availability in Peer-to-Peer Overlay Networks

THESIS

Submitted in partial fulfillment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY

by

K HARI BABU

Under the Supervision of
Prof. Chittaranjan Hota

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI (RAJASTHAN) INDIA

July 2012

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
PILANI, RAJASTHAN

CERTIFICATE

This is to certify that the thesis entitled "**To Develop Algorithms for Efficient and Secure End-to-End Service Availability in Peer-to-Peer Overlay Networks**", submitted by **K Hari Babu** ID.No **2006PHXF016P** for the award of Ph.D. degree of the Institute, embodies original work done by him under my supervision.

Signature of the supervisor : ____________________

Name : **DR. CHITTARANJAN HOTA**
Designation : Associate Professor & HoD,
Department of Computer Science
and Information Systems,
BITS Pilani, Hyderabad Campus
Hyderabad

Date : __________

tad-vak-visargo janatagha-viplavo
yasmin prati-slokam abaddhavaty api
namany anantasya yaso ankitani yat
srnavanti gayanti grnanti sadhavah
Srimad Bhagavatham (1.5.11)

Those teachings aimed at glorifying the Supreme Lord have the potency to bring revolution in the impious lives. Those who are thoroughly honest, hear, sing and deeply accept them into their lives.

Dedicated To

one who is thoroughly honest in hearing, singing and accepting the teachings of Srimad Bhagavatham .....

Acknowledgements

I would like to express my gratitude to several personalities for their constant guidance, help, support and well-wishes.

I acknowledge with great pleasure, my deep sense of gratitude to my supervisor Dr. Chittaranjan Hota for his constant encouragement, valuable guidance, and inspiring suggestions throughout the course of this work. At several times, research went into despair state. At that time, having a few minutes talk with him kindles the hope and lets the work keep rolling. He gave the direction, and clues that made me go deeper into the work. He never let me stagnate but kept me on the move.

I would like to thank my DAC members, Prof. Sundar Shan Balasubramaniam and Prof. Shanta Pendkar for reviewing my proposal and for their valuable inputs throughout the thesis work. I feel indebted to colleagues Prof. Rahul Banerjee and Murali for sharing their views on some of my work and encouraging me. I would like to thank Prof. R K Mittal, Prof. Siva Subramaniam and Prof. B R Natarajan for their constant encouragement and help. I would like to thank Prof. N N Sharma, Prof. Navneet Goyal, Prof. A K Giri, Prof. Raghunathan, and Prof. B K Rout for encouraging me to complete my thesis.

I would like to thank my students Dayakar, Saravana, Dushyanth, Arindam, Bhavik, Anilkumar and Sirish for their sincere project work. I would like to thank Ganeshan Subramanyam, and Amit Raj Narayan for spending time in discussing and suggestions. I would like to thank Prof. Sasu Tarkoma and John Douceur for their valuable comments on some of the work. I thank all the anonymous reviewers who spent their valuable time to read in detail and suggested several ways of improvement. I thank Abhishek Varshney for helping me in converting word documents into Latex.

Thanks are due to Prof. B N Jain, Vice-chancellor and Prof. G. Raghurama, Director for the constant support and concern. I would like to gratefully acknowledge Prof. Ashish Kumar Das, Dean, RCD and many other administrators for providing us with suitable working atmosphere. I thank all my colleagues in our department and those known to me in other departments for their well wishes.

Words fail to express my gratitude to my parents Sri K Pradadesi and Srimati K Syamala for their love, care and patience. Without their love and support i would not have got any education. I thank my sisters Saritha and Jayanthi for their affection.

I deeply thank those who dedicated their lives to Srimad Bhagavatham, the oasis in this deserted age.

Abstract

Peer-to-peer overlay traffic forms a major part of the Internet traffic. Peer-to-peer overlays have prominent applications in file-sharing, communication, and content distribution. Peer-to-peer paradigm is largely applied in file-sharing over internet. More than 54 peta bytes of data is being shared on these networks. Due to their decentralised nature, and large size, searching for a file is an important service. Efficiency of search service affects the overall performance of the overlay. User directly perceives the effectiveness of search. In this work, approaches for improving efficiency and quality of search are developed and evaluated. Another important consideration for decentralised networks is security. In a peer-to-peer overlay, the end systems act as routers. Traffic passes through end-systems. Also end systems contribute to the storage, and computational resources of the network. In this work, we address an important problem known as Sybil attack which can exploit the routing service and resources shared by other honest participants for selfish purposes.

Two basic approaches to search in peer-to-peer overlays are flooding and randomwalk. Flooding has the advantage of quick response and enormous traffic and randomwalk has the advantage of negligible traffic and very delayed response. The in-between approach consists of intelligent selection of neighbours to forward the queries. A content-oriented metric is proposed which proved to have edge over the other metrics. Other metrics are built upon the observations over a period of time. This alone is insufficient to guide the queries. Therefore the proposed approach considers what type of content the neighbours are sharing and how popular it is and compares this with what type of content query is looking for and how popular is the content. This information is subjective and is modelled using Fuzzy Sets. Queries are guided

by the probabilities obtained from fuzzy sets. The proposed approach has got 39% improvement on search efficiency index.

Search without indexing is not imaginable. Floating Indexes technique is proposed to improve search in peer-to-peer networks. This technique doesn't need any extra messages but utilise the query traffic to disseminate and maintain the indexes. This technique has increased the search efficiency metric to 154. When this technique is combined with fuzzy based neighbour selection, the metric has increased to 174. The methods of propagating are thoroughly analysed and concluded that 'breadth-wise dissemination of indexes using randomwalk' approach most utilizes the query traffic to disseminate the indexes.

It is further observed that although search algorithm finds an object in the system, it doesn't have the capability to take user expectations in guiding the search. Since, in most cases, a user searches for a file with a purpose of retrieving it, i.e., either through downloading, or streaming etc, search and retrieval are linked together. Users do have variety of expectations. To increase user satisfaction, a QoS-constrained search is developed. This model takes user expectations into account and transforms them into QoS constraints. Search is performed for the file in such a way that QoS constraints are best matched. A cost metric is designed to guide the user in selecting the most suiting result of the query. Simulations have shown that this algorithm performs according to user expectations without increasing load on network. This search algorithm involves finding a feasible path with multiple constraints. It is compared with SAMCRA and found to be giving exactly the same results for 84% of the queries.

A challenge-response model solution to Sybil attack is developed. It works on challenging the storage constraint of the Sybil nodes. It has the advantage that it doesn't require simultaneous challenges to be issued to all Sybil identities at the same time. Simulations have shown that when 40% of the nodes in the network are Sybils, this approach can detect upto 60% of Sybil identities.

A solution based on Psychometric tests is proposed for detecting Sybil attacks.

It works on finding the personality characteristics of the people behind the virtual identities. Metrics, tests and clustering methods are developed to identify the Sybil groups. Through experiments it has been shown that 75% of the Sybil groups or 69% Sybil identities are detected. It is also shown that the method has the capability to cluster Sybil identities of the same group together so that they all can be issued a challenge simultaneously.

Table of Contents

List of Figures

List of Tables

List of Abbreviations/Symbols

Term	Definition
P2P	Peer-to-Peer
TCP	Tranmission Control protocol
IP	Internet Protocol
RFC	Request For Comments
SMTP	Simple Mail Transfer Protocol
NNTP	Network News Transfer Protocol
MBone	Multicast Backbone
SETI	Search for Extraterrestrial Intelligence
CAN	Content Addressable Network
DHT	Distributed Hash Tables
SHA	Secure Hash Algorithm
TTL	Time to live
GUID	Globally Unique Id
APS	Adaptive Probabilistic Search
RTT	Round-trip Time
NAT	Network Address Translation
SE	Search Efficiency
GT-ITM	Georgia Tech Internetwork Topology Model
fpp	false positive probability
FI	Floating Indexes
FIB	Floating Indexes Breadth-wise
FID	Floating Indexes Depth-wise
QIR	Query Index Record
NIR	Node Index Record
ABF	Attenuated Bloom Filters
IETF	Internet Engineering Task Force
QoS	Quality of Service
RSVP	Resource Reservation Setup Protocol
MCOP	Multi-constrained Optimal Path
MCDM	Multi-criterion Decision Making
SAMCRA	Self-Adaptive Multiple Constraints Routing Algorithm
MBTI	Myers-Briggs Type Indicator
d	Average node degree
h	Number of hops

Chapter 1

Introduction

The peer-to-peer (P2P) paradigm started becoming popular in the middle of 2000 among the music lovers. Since then, due to its inherent positive characteristics, the term 'P2P' has become very popular amongst Internet users, researchers and industries. The emergence of P2P file sharing networks has increased the interest amongst Internet users to use the Internet beyond web browsing and exchanging e-mails. There is large number of applications such as Napster [nap 1999], Gnutella[gnu 2000], Kazaa [kaz 2001], eDonkey [edo 2000] developed for deploying P2P technologies in the Internet. There is an exponential increase in number of users taking interest in these applications. Researchers across the world have taken great interest in P2P networks and contributed to their viability in several aspects. Researchers have contributed large number of models based on P2P overlay networks, suiting to different requirements such as data sharing, distributed file systems, anonymity, media streaming etc. Due to their ease of deployment and scalability, industries have formed the consortium named "P2P-Next Generation" [p2p 2008] to support research to effectively use the P2P technology for mass media distribution. P2P traffic occupies a major chunk of the Internet traffic [Saroiu *et al.* 2002a]. Table 1.1 shows the huge increase in traffic generated by peer-to-peer protocols [cac 2005]. Figure 1.1 shows the dominent proportion of traffic by peer-to-peer protocols in all regions of the world [ipo 2010]. The kind of impact they generate on the whole Internet motivates researchers to study and improve the scalability and performance of these networks.

Table 1.1: **P2P Traffic** - Percent of Internet traffic occupied by P2P Protocols

Year	Web Traffic	FTP Traffic	P2P Traffic
1999	65%	10%	-
2005	24%	2%	72%

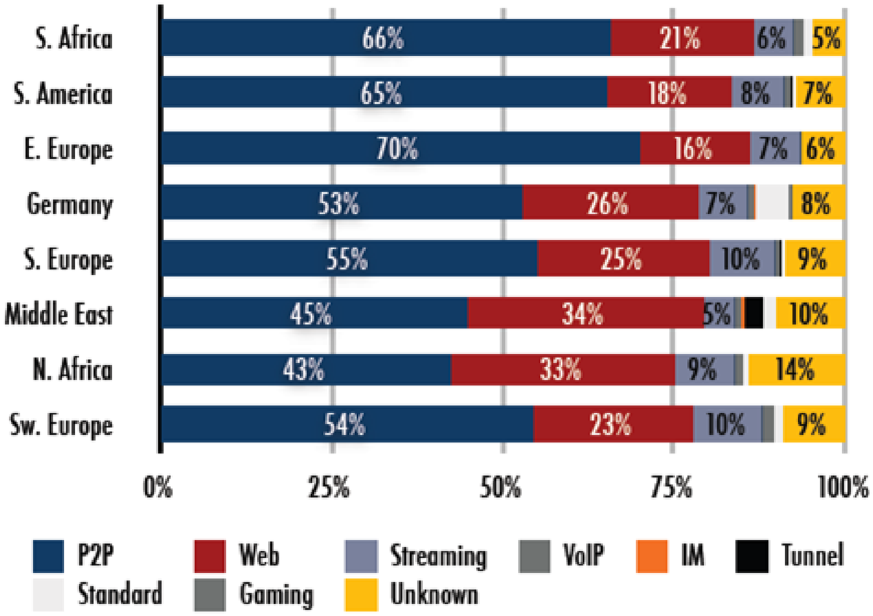


Figure 1.1: Distribution of protocol traffic in 2008/2009 - source:[ipo 2010]

1.1 Defining Peer-to-Peer Computing

Peer-to-peer is a very broad term that can accommodate any two systems communicating at the same level. In particular, this is true for any two systems communicating on the Internet at TCP layer and IP layer. A broader definition that is widely accepted is "*peer-to-peer is a class of applications that take advantage of resources-storage, cycles, content, human presence-available at the edges of the internet*" [Oram 2001]. This definition includes not only fully decentralized peer-to-peer systems but also Napster [nap 1999], SETI@home [set 1999], instant messaging systems yahoo, gTalk, MSN, AOL which are supported by central servers for their operation but appear to be using resources on end systems just like peer-to-peer applications. This definition also includes applications from the field of Grid computing which are to provide support for sharing resources on systems situated geographically apart.

[Androutsellis-theotokis & Spinellis 2004] gives a stricter definition of peer-to-peer computing. In this definition, two characteristics were identified that define peer-to-peer computing. The first characteristic is that nodes in the peer-to-peer network share the resources by direct exchange without the mediation of a centralized server. Since the nodes in the peer-to-peer network need to communicate with each other without a central coordinating server, the nodes themselves have to actively participate in carrying out the tasks that enable communications in the network. That means the nodes have to get involved in routing messages and information, executing network maintenance routines such as to update the routing state, searching for content on behalf of others, replicating objects as per network policies, etc. This characteristic of peer-to-peer networks doesn't match with Napster which uses a centralized server to index clients and their data. The peers are sharing resources but they don't get involved in managing the network. In Napster[nap 1999], there is no network formation from the view of a peer. Each client views only a central server. The second characteristic is that nodes in a peer-to-peer network are capable of dealing with instability and variable connectivity as the norm and they automatically adapt to highly transient population and failures in the network. That means the network topology is dynamically changed by participating nodes so as to maintain connectivity and performance. This leads to the definition: "*Peer-to-peer systems are dis-*

tributed 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" [Androutsellis-theotokis & Spinellis 2004]. This definition includes the super-peer based peer-to-peer architectures where some high capacity peers are dynamically chosen to act as a super-peer. Super peers are not the same as centralized servers, for they are peers of the network dynamically chosen and they leave at any time. This doesn't limit the scalability of super-peer network as in the case of centralized servers. Some peer-to-peer systems do use central servers for tasks such as authentication, and boot-strapping but such servers don't get involved in the network operations.

RFC 5694 [Camarillo & IAB 2009] gives an official definition of a peer-to-peer system: A system is to be considered P2P if the elements that form the system share their resources in order to provide the service the system has been designed to provide. The elements in the system both provide services to other elements and request services from other elements. This definition highlights the two characteristics i.e. (i) peers must share their resources for the purpose of network services and (ii) peers must be both service requesters and service providers.

1.2 Peer-to-Peer and Grid Computing

Grid computing is emerging out of the scientific and academic area in to the commercial world. Current grid computing systems use client-server architecture for distributed computing. The Grid computing is defined by [Foster *et al.* 2001] as "Grids are distributed systems that enable the large-scale coordinated use and sharing of geographically distributed resources, based on persistent, standards based service infrastructures, often with a high-performance orientation". Grid computing, like peer-to-peer computing, is also about sharing resources. But the sharing that grid computing is concerned with is not primarily file exchange, but rather direct access to computers, software, data and other resources. Resources in the grid can be high performance supercomputers, massive storage space, sensors, satellites, software applications, and data belonging to different institutions. The grid provides the infrastructure that enables institutions like commercial companies, universities, government institutions, and laboratories to form virtual organisations that share resources and collaborate for the sake of solving common problems [Foster & Kesselman 2004]. The kind of issues addressed by grid computing includes authentication, authorization, resource discovery and resource access etc. The initial peer-to-peer applications targeted file sharing without any access control.

As the systems size grows, grid computing faces the problem of scalability due to its centralized-server approach. Here is the opportunity to apply the techniques of peer-to-peer networks [Marzolla *et al.* 2007] to improve upon scalability.

1.3 Definition of Peer-to-Peer Overlay Networks

An overlay network is a logical (virtual) network at the application layer providing connectivity, routing and messaging amongst the addressable end points [Buford *et al.* 2009]. It has its own own topology different from the underlying physical network. Overlay networks have their way of routing messages with the help of the Internet. Overlay networks are frequently used as a substrate for deploying new network services, or for providing a routing topology not available from the underlying physical network as shown in figure 1.2.

Many peer-to-peer systems are overlay networks that run on top of the Internet. Even before peer-to-peer overlay networks have evolved, overlay networks existed on the Internet for various services. Some of the traditional Internet overlays are listed in Table 1.2 which interconnect infrastructure servers. Here the address space is not virtualized, and churn (joining and leaving of nodes) is not a major design concern. On the other hand, peer-to-peer overlays interconnect end systems (peers), use virtual address space and churn is a major design issue in them.

1.4 Characteristics of Peer-to-Peer Overlay Networks

Peer-to-peer overlay networks have several characteristics that make them very interesting and applicable for practical purposes:

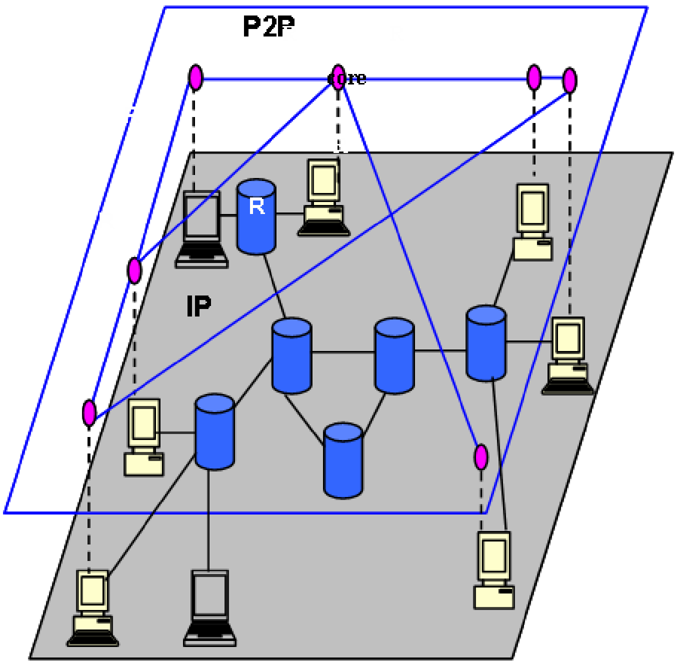


Figure 1.2: Peer-to-peer overlay - logical network built on top of Internet

Table 1.2: Traditional Overlays - overlays that existed prior to P2P in the Internet

Service	Example	Since
Email	SMTP	1970s
Internet news	NNTP	1986
Multicast	MBone	1992
Web caching	Internet cache protocol	1995
Content delivery network	Akamai	1999

Resource sharing: Each peer contributes system resources to the operation of the peer-to-peer overlay. The resources are CPU cycles, memory, storage, content etc.

Networked: All nodes are interconnected with other nodes in the peer-to-peer overlay i.e., any node is reachable from any other node. The nodes form a connected graph. It is the collective responsibility of the nodes to see to it that the network is not partitioned.

Decentralization: The behavior of the peer-to-peer overlay is determined by the collective actions of peer nodes, and there is no central control point. This is the difference between client-server applications and peer-to-peer applications.

Symmetry: Nodes assume equal roles in the operation of the peer-to-peer overlay network. In many overlays such as Kazaa [kaz 2001] this property is relaxed. In such networks, some peers are dynamically chosen to be super-peers. Only super peers act as service requesters and service providers.

Autonomy: Participation of the peer in the peer-to-peer overlay network is determined locally.

Self-organization: Organizing the nodes into topologies as nodes join and leave and adapting to failures are also properties of P2P overlay networks. Self-organization has two-fold objective: self-maintenance and self-repair. For self-maintenance, node joining and leaving is handled in a distributed manner, without requiring the member change information to be propagated through the entire network. When a node fails without a warning, its connected nodes still regard it as alive. In this case, a self-repair function need to be run regularly to keep the network connected.

Scalable: This is a crucial characteristic that makes peer-to-peer an attractive alternative to client-server architectures. To operate peer-to-peer overlays with millions of simultaneous nodes, scalability is prerequisite. That means the resources consumed at each peer grow as a function of overlay size which is less than linear. It also means that response time of the network services doesn't grow more than linearly as a function of overlay size.

Stability: Within a maximum churn rate, the peer-to-peer system should be stable, i.e., it should maintain its connected graph and be able to route deterministically within a practical hop-count bound.

1.5 Applications of Peer-to-Peer Overlay Networks

Peer-to-peer overlays have been recognized for their applicability in various domains. A variety of applications have been developed based on peer-to-peer overlays

1.5.1 Distributed computing

To utilize the computing power available at the edge nodes, SETI@home[set 1999] project has divided its work into small units and distributed them to the peers or end user devices. This is a very simple example of applying peer-to-peer paradigm in solving larger problems.

1.5.2 Internet Services

Peer-to-peer applications have been developed to support variety of internet services. IP level multicasting is costly and limited in the Internet. Scribe, a peer-to-peer overlay application is developed by [Castro *et al.* 2002b] to support large-scale multicasting in the Internet. I3 is developed by [Stoica *et al.* 2002] providing same service offered by DNS but using a completely decentralized infrastructure. This doesn't need root servers. Security related issues such as intrusion detection [Janakiraman *et al.* 2003], and spam [Zhou *et al.* 2003] are also handled by peer-to-peer overlay based applications. [Keromytis *et al.* 2002] have developed secure peer-to-peer overlays to provide protection against denial of service and virus attacks. Squirrel [Iyer *et al.* 2002] facilitates mutual sharing of web data objects among client peers, and enables the peers to export their local caches to other peers in the network, thus creating a large shared virtual web cache.

1.5.3 Distributed Databases

Several researchers have done work on designing distributed database systems based on peer-to-peer overlay networks. Local Relational Model (LRM) is a model proposed by [Bernstein *et al.* 2002] to support databases in peer-to-peer environment. PIER is a scalable distributed query engine developed by [Huebsch 2008] to support relational queries on data that is spread across thousands of peers in a peer-to-peer overlay network.

1.5.4 Communication and Collaboration

Jabber as explained by [Saint-Andre 2005] is an XML-based application developed to provide messaging and presence. Skype [sky 2003] is a peer-to-peer application that provides voice and video calls, voice calls to PSTN end points, and instant messaging. It connects around 15 million users simultaneously over the globe. Its characteristics are analysed in [Xie & Yang 2007]. Groove [Edwards 2002] provides collaborative services such as group calendaring, collaborative editing and drawing, and collaborative web browsing among its users. SharedMind [Ang & Datta 2010] is a collaborative peer-to-peer application that supports editing of shared data using mind-maps.

1.5.5 Content Streaming and Multi-casting

Digital content on the Internet, its access by users and streaming of such content is growing exponentially each day demanding bandwidth at servers and internet backbone. Content Delivery Networks (CDN) are facing problems with bandwidth in streaming video content. Many CDN networks are using peer-to-peer overlays to stream the content in a distributed way. Commercial systems such as Kontiki [kon 2000], Zattoo [zat 2006] provide mass media distribution over internet using peer-to-peer overlays or P2PTV. Various models are used, including torrent-style distribution, application layer multicasting, and hybrid CDNs. Example P2PTV applications include Babelgum [bab 2005], Joost [joo 2007], PPTV[PPT 2005], PPStream [PPS 2010], SopCast [sop 2004]. P2PTV is expected to play an important role in future IPTV deployments. Narada[hua Chu *et al.* 2002], Scribe[Castro *et al.* 2002a], Management overlay network (MON)[Liang *et al.* 2005] are some peer-to-peer applications that support group management and multicasting services.

1.5.6 Content Publishing and Storage

Applications are developed to provide content management (updating, removing, version control). Users will be able to publish, store, and distribute content in a secure and persistent manner. OceanStore[Kubiatowicz *et al.* 2000], is global-scale, highly available storage utility. It is built on Tapestry to disseminate encoded file block, efficiently and clients can quickly locate and retrieve near by blocks by their Id, despite server and network facilities. PAST [Rowstron & Druschel 2001] is a large-scale persistent storage utility that is based on Pastry. Each peer is capable of initiating and routing client requests. Pastiche [Cox *et al.* 2002] is a simple and inexpensive backup system built on Pastry that provides content-based indexing and convergent encryption.

1.5.7 File Sharing

File sharing is the most popular application in peer-to-peer overlay networks. Content sharing systems offer mechanisms for content search and for transferring content. Content transfer can happen in two ways: downloading and streaming. Streaming means a node consumes the content while it is being transported. Downloading means the content is fully transferred to local node and then it is used. File sharing systems generally support downloading of content. In this thesis work, the focus is on file sharing peer-to-peer overlay networks. There are several examples of file-sharing applications. Gnutella, Free haven, Kazaa, Napster, Bittorrent[bit 2001], eDonkey are some of the well known examples. Table 1.3 lists applications developed over peer-to-peer overlays using different architectures. Table 1.4 lists some of the peer-to-peer applications which are implemented by commercial organizations.

1.6 Classification of P2P Overlays by Degree of Centralization

Peer-to-peer overlay networks by their characteristic are expected to be fully decentralized. But in practice this is not always true. The following three kinds of centralization are found among the current peer-to-peer systems.

1.6.1 Purely Decentralized Architectures

All peers in the network take the same roles i.e. there is complete symmetry. Each peer acts as a client and also as a server. Peers in such systems are known as "servents". All peers participate in the network operations such as routing the message, data, searching for file requests etc. These networks exhibit robustness to churn and node failures. Nodes have limited knowledge about the network. Therefore, they can’t provide guarantee on search efficiency and content availability. Gnutella 0.4, FreeHaven, Freenet, Chord, Pastry,

Table 1.3: Peer-to-Peer Applications - classified according to architecture of P2P overlay

Application Purpose	**Purely Decentralized**	**Partially Decentralized**	**Hybrid Decentralized**
Content Publishing & Storage	Freenet, PAST, OceanStore, CFS		
File Sharing	Gnutella, Free Haven	Gnutella 0.6, Kazaa	Napster, BitTorrent
Anonymous Storage	Freenet		
Communication & Messaging	Groove, SharedMind	Skype	Jabber
Backup operations	Pastiche		
Infrastructure base	Chord, Pastry, Tapestry, CAN		
Routing	Resilient Overlay Networks (RON)		
Search engines	Minerva, ODISSEA		
Computation			SETI@home
Web Caching	Squirrel		
Multicasting	Split Stream, Scribe, Bayeux		
Spam filtering	SpamWatch		
Domain name lookup	DNS using Chord		
Multimedia Streaming	PPLive, Coolstreaming		
Intrusion Detection	Indra		

Table 1.4: Commercial P2P Overlays - P2P overlay networks formed and maintained by commercial organizations

Application	**Purpose**	**Organization**	**Since**
Kontiki	Hybrid peer-to-peer content distribution targeting media content	M K Capital	2008
Skype	Voice over peer-to-peer	eBay	2006
Groove	Collaboration without any help of central server. Known customers: Dell, Department of Defense, GlaxoSmithKline and Veridian	Groove Networks	2001
Magi	P2P infrastructure platform for building secure, cross-platform, collaborative applications	Endeavours Technologies	2001
Zattoo	Streaming to TV viewers over internet	Zattoo	2006
CoolStreaming	To share television content using swarms	Roxbeam Corp.	2005
PPTV	Live and on-demand TV service over internet	PPLive	2005
Joost	Streaming legal videos in agreement with publishers	Adconion Media Group	2009
Alluvium	For broadcasting over internet	Foundation for Decentralization Research	2003
RawFlow	internet broadcasting of audio and video	RawFlow Ltd.	2002

Tapestry are examples of this architecture. Topology of Purely Decentralized Architecture is shown in Figure 1.3.

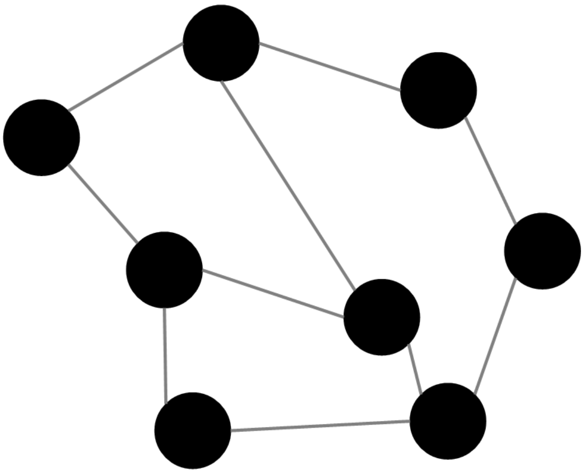


Figure 1.3: Purely Decentralized P2P overlay - all nodes in the network take the same roles

1.6.2 Partially Decentralized Architectures

These systems are also same as purely decentralized architectures except that some nodes take a more important role. Such nodes are known as "super peers" or "ultra peers". These super peers collect and store file indexes of all the local peers attached to it. All super peers are connected like in a pure decentralized architecture. Super peers don't pose the problem of single point of failure, as they are chosen dynamically. If some super peer leaves or fails, local peers automatically connect to another super peer and the network will chose a replacement for the failed super peer. Gnutella2, Kazaa, eDonkey2000, Edutella[Nejdl *et al.* 2002] are examples of this architecture. Topology of Partially Decentralized Architecture is shown in Figure 1.4.

1.6.3 Hybrid Decentralized Architectures

In this architecture, there is a centralized arrangement to facilitate the interactions between peers. Central servers maintain metadata such as file indexes or data segment locations etc. Surely these servers become bottlenecks and single point of failures. Napster, Publius [Waldman *et al.* 2000], BitTorrent are examples of such architectures. Topology of Hybrid Decentralized Architecture is shown in Figure 1.5.

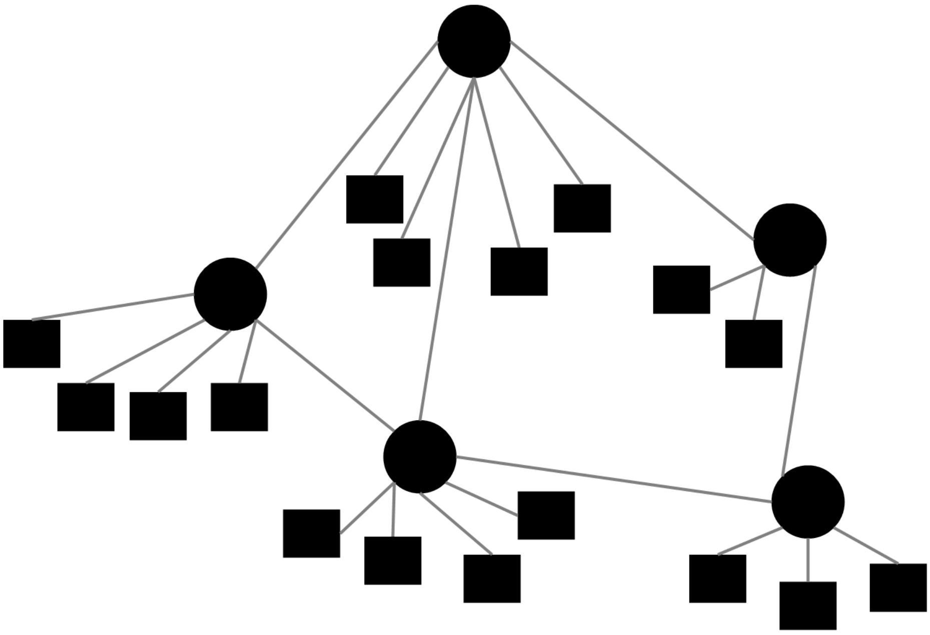


Figure 1.4: Partailly Decentralized P2P overlay - Some nodes are attached to ′super peers′ which perform important roles

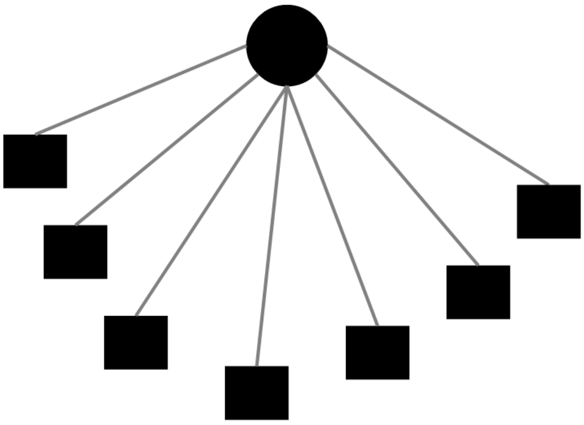


Figure 1.5: Hybrid Decentralized P2P overlay - Nodes are attached to central server which facilitates interaction among nodes

1.7 Classification of P2P Overlays by Network Structure

Network structure refers to whether the network topology formation and object placement are based on certain rule or it is left the way nodes want to do it. According to this, there are three categories of peer-to-peer overlay networks as described below:

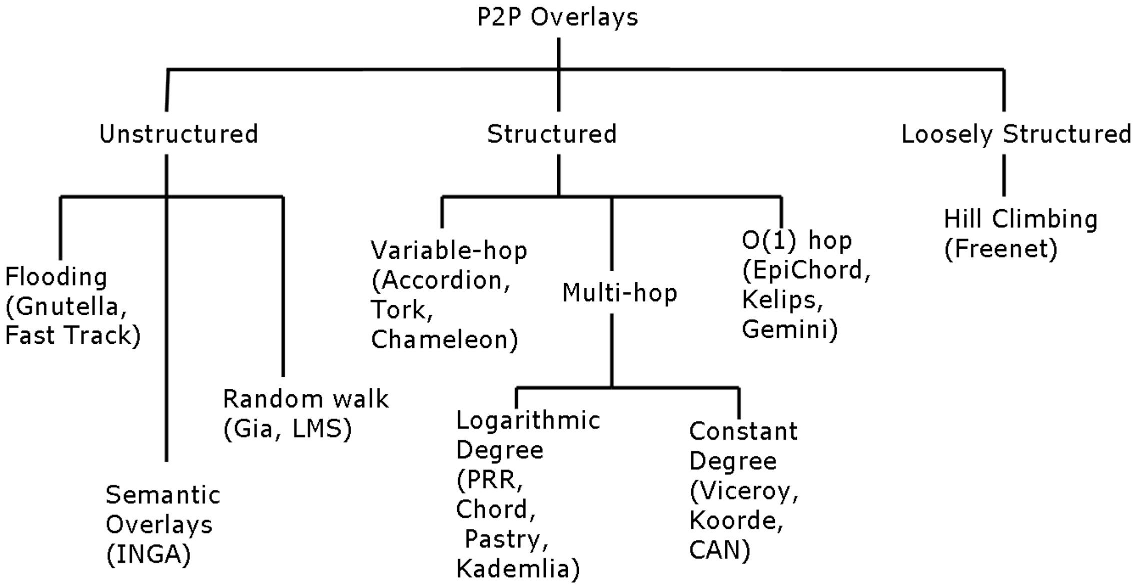

P2P Overlays
Unstructured
Structured
Loosely Structured
Flooding (Gnutella, Fast Track)
Random walk (Gia, LMS)
Semantic Overlays (INGA)
Variable-hop (Accordion, Tork, Chameleon)
Multi-hop
O(1) hop (EpiChord, Kelips, Gemini)
Logarithmic Degree (PRR, Chord, Pastry, Kademlia)
Constant Degree (Viceroy, Koorde, CAN)
Hill Climbing (Freenet)

Figure 1.6: P2P Overlay classification by Overlay Structure - variation in topology and object placement policies

1.7.1 Unstructured Networks

Overlay network is formed without any predefined rules. Nodes and content are added to the network without strict rules. Such overlays form topologies which can be compared to random graphs, scale-free or power-law random graphs, graphs exhibiting small world phenomena and social networks. The placement of content (files) is completely unrelated to the overlay topology. Therefore, content needs to be located through messaging. Peer relies only on its adjacent peers for delivery of messages to other peers in the overlay. Figure 1.6 lists examples.

1.7.2 Structured Networks

Structured networks are proposed as an improvement to unstructured networks. Work on distributed hash tables(DHT) by [Devine 1993] as well as [Litwin *et al.* 1993], [Litwin *et al.* 1996] triggered many researchers to build overlays using DHTs. Plaxton, Rajaraman, and Richa (PRR) [Plaxton *et al.* 1997] presented the first algorithms for distributed object location and routing, using a suffix forwarding scheme. This algorithm was the basis for subsequent influential designs such as Tapestry and Pastry.

In structured P2P overlay networks, network topology is tightly controlled and content is placed not at random peers but at specified locations that will make subsequent queries more efficient. The content is distributed among the nodes. The nodes and objects are mapped to a key space. The key space of the objects is partitioned and distributed among the peers. They provide deterministic time bounds on lookup. As shown in Figure 1.6, there are different types of structured networks namely logarithmic degree, constant degree, variable hop, $O(1) - hop$ etc. Logarithmic degree means each node has to maintain links with neighbours in the logarithmic order of overlay size. Constant degree means each node maintains the same number of neighbour links irrespective of overlay size. Variable-hop indicates that the lookup of the object will take variable number of hops depending on the node capacity and network capacity. $O(1) - hop$ means that each node keeps the references to all other nodes in the overlay. Table 1.5 shows the example infrastructures and applications using structured networks.

1.7.3 Loosely Structured Networks

In these networks, structure evolves over a period of time. There are no predefined rules that specify the node positions within the overlay. Based on the redundant replication of the files, the nodes accumulate references to other nodes. Since these references govern the overlay structure, the structure evolves with content placement. Freenet [Clarke *et al.* 2001] has this type of overlay network.

1.8 Search in Peer-to-Peer Overlays

Peer-to-peer file sharing systems lack a centralized index placed at a group of server machines. The index is placed at peers themselves i.e. it is a distributed index. Therefore, search in a peer-to-peer overlay refers to finding any given data item by looking at the indexes stored at individual peers. Since it is a distributed algorithm, the performance of the

Table 1.5: P2P Overlays by Structure and by Decentralization - combining both type of classifications

Network Structure	**Decentralization**		
	Hybrid	**Partial**	**Pure**
Unstructured	Napster, Publius	Kazaa, Gnutella, Edutella	Gnutella, Free-Haven
Loosely structured			Freenet
Structured Infrastructures		Structured SuperPeers	Chord, CAN, Tapestry, Pastry, Kademlia , Viceroy
Structured Systems			OceanStore, Mnemosyne, PAST etc

algorithm depends on overlay geometry and routing protocol. Search becomes complex in peer-to-peer file sharing systems because of transient population of nodes and dynamic changes in the content. The centralized search systems such as Google, Yahoo, etc. have to deal with web pages which live longer than objects in the peer-to-peer file sharing systems. Such centralized systems don't prove to be efficient in large-scale distributed systems due to content and node population dynamism. Therefore search in peer-to-peer file sharing systems has different requirements when compared to a centralized search mechanism.

1.8.1 Design Characteristics of a Search Mechanism

The following gives important requirements for a search algorithm in large-scale peer-to-peer file sharing overlays.

1.8.1.1 Decentralization

There is always a trade-off between the scope of the index and maintenance cost. In one extreme every node can keep index of all the objects available in the network. That will surely make all the queries one-hop. But the cost associated with maintaining such an index prohibits such an approach. On the other extreme, searching loses its grace without indexing. Indexing is the most important tool for searching [Baeza-Yates & Ribeiro-Neto 1999]. The index construction process should be distributed among the participating nodes. The index itself should be distributed among the overlay nodes for achieving uniform load distribution and fault-resilience.

1.8.1.2 Efficiency

While building the index and while looking it up, the algorithm should not consume significant resources of the network namely the bandwidth and storage at each node. The high rate of introducing new files into the network and relocation of documents either due to replication or rejoining of a node makes the index building and maintaining process very intensive.

1.8.1.3 Scalability

Efficiency of the search mechanism should not degrade with increase in the network size.

1.8.1.4 High Quality Results

The quality of search is measured by its recall rate, precision and response time. Recall rate is measured as the percentage of files that are returned against the actual number of such files available in the network. Precision refers to the number of results that are relevant to the query. Search algorithm is expected to have high completeness, high precision and fast response time.

1.8.1.5 Fault-resilience

Peer-to-peer overlays are subjected to churn i.e., high rate of joining and departing nodes. Nodes depart from network without prior notification. The search algorithm is expected to adapt to such failures providing quality of service efficiently to the user.

1.8.2 Search Performance Metrics

Search performance can be measured in the following view points. These are explained in [Daswani *et al.* 2002].

1.8.2.1 Efficiency

Peer-to-peer networks use the computation, storage and bandwidth resources of peer nodes. Efficiency of search algorithm is measured in terms of consumption of these resources. In overlays, the bandwidth consumption is approximately proportional to number of messages required to complete the search. The storage refers to the size of the indexes the peers have to keep with them. The computation refers to the complexity of the operations the individual nodes have to carry out in search operation.

1.8.2.2 Quality of Service (QoS)

Quality of service (QoS) focuses on user perceived qualities where as efficiency refers to the utilization of overlay network resources. QoS depends on individual application. In general, it can be measured as number of results, response time, and relevance. Response time is measured as the time elapsed from the time when the query is fired and till the time when the first reply is received. Relevance is measured by recall rate and precision.

1.8.2.3 Robustness

QoS and efficiency have to be tested against the overlay dynamics. The node population as well as the content is variably changing. This directly affects the search performance. The metrics mentioned above are observed for their change with respect to nodes join and leave and content additions and changes to the network.

1.8.3 Components in a Distributed Search Mechanism

A distributed search mechanism, in large-scale peer-to-peer overlays, is composed of three components namely query semantics, translating semantics for routing, and routing the query as shown in figure 1.7.

Peers share different types of objects in a file sharing network. The meta information consisting of behavioural and functional aspects of the object is expressed in terms of properties in a schema. In file-sharing systems, these properties for each type of object such as song, movie, software etc. are globally known in standard schemas. Expressiveness refers to the capability of search in expressing the user expectations. The minimum

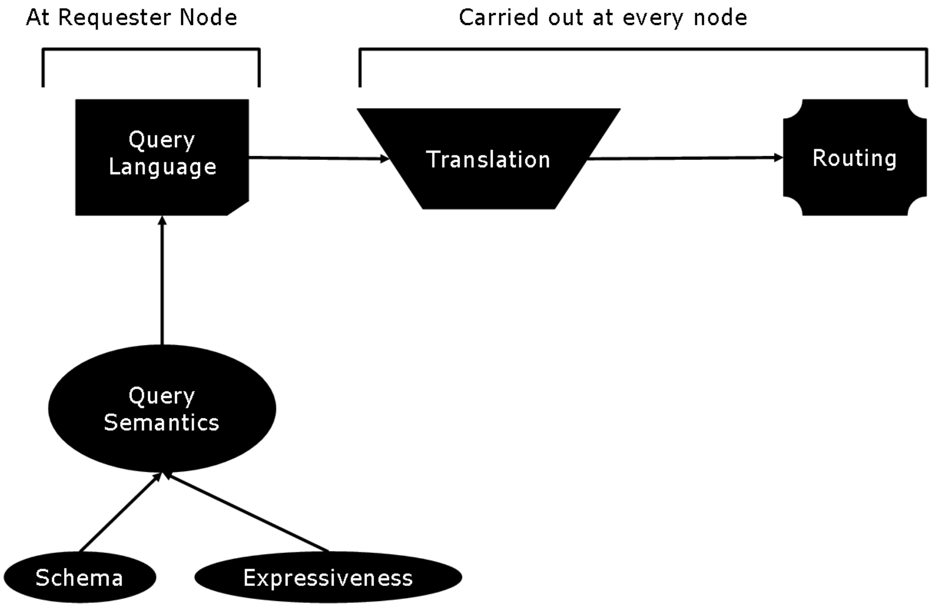


Figure 1.7: Components of decentralized search mechanism - some components are carried out at the requester node and the other at every node receiving the search query

level of expressiveness is exact-keyword-match which is supported in DHT[Devine 1993] based structured networks. That means the user has to specify exact properties that were used to hash the file and generate its key. User has to know the properties of the file completely. Another level of expressiveness is partial-keyword-match. This is supported in unstructured file-sharing systems. Query semantic refers to formation of the query as expressed by user and supported by the network using standard schema. Query is specified in a query language. In most of the file sharing networks, there is no specific language used. To support complex queries involving logical, and relational operators, XQuery[Chamberlin & Robie 2007], XPath[Kay & Robie 2007], SPARQL[Prud'hommeaux & Seaborne 2007] are proposed.

The translation component transforms the query information into information useful for routing purpose. In unstructured networks only when informed search schemes are used, the query semantics are transformed to calculate some metrics to enhance the search efficiency. These metrics are used in deciding where to forward the query. In blind search schemes, the routing is basically either random or a broadcast. In structured networks based on DHT, the routing is based on keys. The keys are transformation of query semantics using a hashing scheme. The translation component transforms the query semantics into a key.

Routing component involves forwarding the query to one or more neighbours at every node. There is content-based routing and key-based routing. In content-based routing, the query semantics are utilized for making routing decisions at every hop. Content-routing allows partial match queries and complex queries. It doesn't guarantee search completeness or discovery of rare objects. In structured networks, the routing is carried out using routing tables maintained at each node. The key based routing is efficient but doesn't support partial match and complex queries.

In unstructured networks, the efficiency of search mechanism depends largely on routing component. Routing policy affects the bandwidth of the network. The computation overhead at each node is affected by translation component. The query semantics will help in enhancing the efficiency and QoS of the search mechanism.

1.9 Peer-to-Peer Overlay Networks: Examples

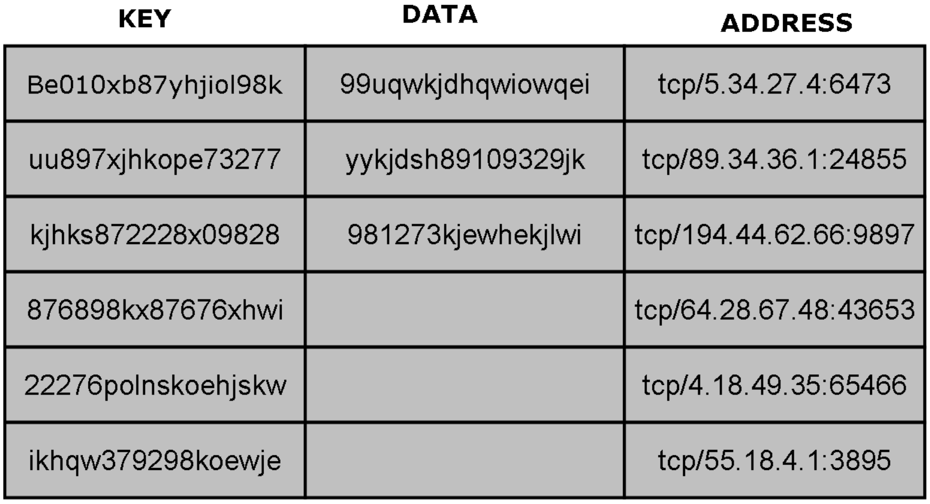
This section presents some widely used or deployed overlay networks from the point of view of their search mechanisms.

1.9.1 Freenet

Freenet is a distributed peer-to-peer system designed for anonymous storage. The anonymity is in terms of who is storing the file, what the contents of the file are, who is requesting the file, and who is fulfilling the request. It offers robustness, scalability, efficiency and privacy. Freenet is purely decentralized system. Each node in the network allocates disk space to be used by Freenet. Freenet provides a virtual file system logically connecting disk space provided by each node in the network.

Each data item is associated with a key. Node stores the data item along with the key. Every node in the network maintains references to some other nodes in the network. These references are accumulated based on file exchange over a period of time. When DataInsert message is sent along with the file, the file is cached on all nodes on the path. This caching will help in load balancing. Every node on the path will add the reference of the node from which it has received the file. The same is the case when DataRequest message is replied. Therefore the lists of references are dynamic. Depending on Least

Recently Used (LRU) policy, nodes keep deleting the references to accommodate the new references in a constant-sized routing table. The routing of messages for inserting and retrieving files is carried out using these references. Since the routing of messages is based on links (references) that have already been formed based on keys associated with files, there is locality on key space. Similar keys are grouped together. A sample routing table (stack) is shown in figure 1.8. Routing happens by proximity in the keys. The local node decides which node will receive the next query by the closeness of the destination key with the list of keys it has. The locality in key distribution helps in scalability of the network.



KEY	DATA	ADDRESS
Be010xb87yhjiol98k	99uqwkjdhqwiowqei	tcp/5.34.27.4:6473
uu897xjhkope73277	yykjdsh89109329jk	tcp/89.34.36.1:24855
kjhks872228x09828	981273kjewhekjlwi	tcp/194.44.62.66:9897
876898kx87676xhwi		tcp/64.28.67.48:43653
22276polnskoehjskw		tcp/4.18.49.35:65466
ikhqw379298koewje		tcp/55.18.4.1:3895

Figure 1.8: Stack used by a Freenet node - source: [Oram 2001]

Since Freenet is a fully decentralized network, there is a possibility of malicious nodes returning the false documents. Since there is inherent caching of the objects over the path, the false documents can spread in the network like a wildfire. To prevent this Freenet uses keys. It offers three types of keys. A document can be associated with one of these keys. But other type of keys can be used to form a chain of keys to reach the document. Content Hash Keys (CHK) are the result of hashing the file using SHA-1. They are useful in verifying the integrity of the file returned. Keyword Signed Keys (KSK) are based on public key system derived from text strings provided by the user. Signature Verification Keys (SVK) are also based on public key system but are purely binary in nature. SVK provides namespaces using pseudonyms to avoid revealing the identity of the person.

1.9.2 Gnutella

Gnutella is a purely decentralized file-sharing peer-to-peer overlay network. It came into existence in mid 2000 initially with a purpose of enabling users to share their recipes. Later, it became a very popular file-sharing network connecting millions of nodes or users.

In Gnutella, every node is connected to a set of other nodes which are called as neighbour nodes. Each node specifies files to be shared on the network. To search for a file, a node sends a query to all neighbours. They check with themselves a match. If the match is found or not, the query is forwarded further to its neighbours. If the match is found, a reply is sent to the immediate neighbour from whom this request is received. This way the reply is routed back to the request node. This process is depicted in figure 1.9. The requester node receives several replies.

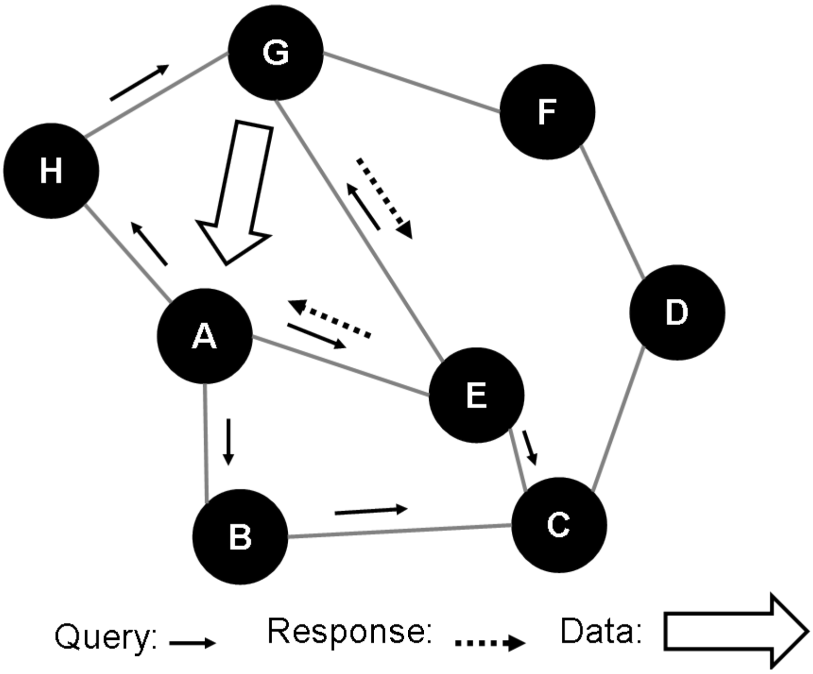


Figure 1.9: Search in Gnutella - Query with TTL=2 fired by node *A* and response is triggered by node *G*. File transfer is through a direct TCP connection from *A* to *G*.

This search algorithm is known as flooding. In flooding, the messages created for the query increase exponentially at every hop. To limit such message explosion, a TTL (time-to-live) limit is defined for the query by the requester. At every node, the TTL is reduced by 1. If a node receives a query with TTL 0, it doesn't forward the query further. Another characteristic of this algorithm is that it avoids looping of messages by assigning a

Globally Unique Identifier (GUID) using algorithm defined in [Leach *et al.* 2005] for every new query. This GUID is associated with that query and its replies all throughout the network. Once they receive the query, nodes cache this GUID and address of neighbour who sent the query. This helps in preventing duplicate queries being forwarded. If the same query is received later from another neighbour, node can detect the duplication and thus avoid forwarding it again as shown in figure 1.10.

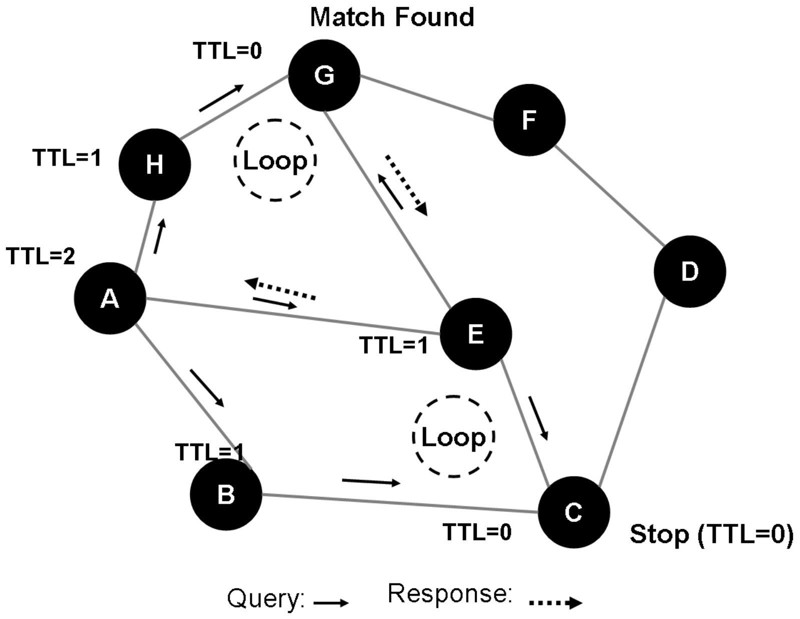


Figure 1.10: Usage of TTL and loop-free condition in Gnutella - Query with TTL=2 fired by node *A* and response is triggered by node *G*. At node *C* and node *G*, GUID prevents duplicate query forwarding.

The GUID also helps in dynamically routing the replies back to the requester without knowing requester identity. This provides requester anonymity in Gnutella. This anonymity doesn't remain if the requester needs to download the file. The downloading of the file from the node which has sent the reply happens by direct TCP connection. In that case requester's identity is revealed.

Gnutella protocol uses five types of messages for its operation: Ping, Pong, Push, Query, QueryHit. Every message has the GUID, message type, TTL, hops and payload length common to them. Each message type is meant for a particular network operation as mentioned in table 1.6.

Table 1.6: Message types in Gnutella 0.4 protocol - each message has its own message format

Message Type	**Description**
Ping	Used for topology discovery of the network. A node receiving a Ping message is expected to respond with one or more Pong messages.
Pong	The response to a Ping message. Includes the address of a connected Gnutella node and information regarding the amount of data it is making available to the network.
Query	The primary mechanism for searching the distributed network. Query message includes minimum speed acceptable and the search keywords. A node receiving a Query message will respond with a QueryHit message if a match is found against its local dataset.
QueryHit	The response to a Query. This message provides the recipient with enough information like IP, port, speed, and list of matching files. Node makes a TCP connection to acquire the data matching the corresponding query.
Push	A mechanism that allows nodes behind firewall to contribute a file to the Gnutella network. If requester can't connect, it sends a "push" message instead, with its IP address and port number. Offerer does an outbound connect to that host, and sends the file.

1.9.2.1 Gnutella Topology Characteristics

Gntella topology is a random graph. A simulated diagram of Gnutella topology is shown in the figure 1.11. There are two characteristics of the graph that help us understand topology of the network. They are listed in table 1.7 for regular and random graphs.

Table 1.7: Network topology graph characteristics - for a graph with n vertices and average number of edges per vertex is k. l indicates the actual number of edges in the graph.

Graph type	**Pathlength**	**Cluster coefficient**
Regular Graph	$\frac{n}{2k}$	$\frac{l}{\frac{k(k-1)}{2}}$
Random Graph	$\approx \frac{\log n}{\log k}$	$\approx \frac{k}{n}$

Characteristic pathlength is defined as the average of shortest paths of all vertex pairs in the graph. If this metric is small, it means that any two nodes in the network can communicate with in small number of hops. The message routing cost is low. Another characteristic is clustering coefficient, which is defined as proportion of actual number of links to the number of all possible links among neighbours of a node. The characteristic clustering coefficient is the average of clustering coefficients of all vertices in the graph.

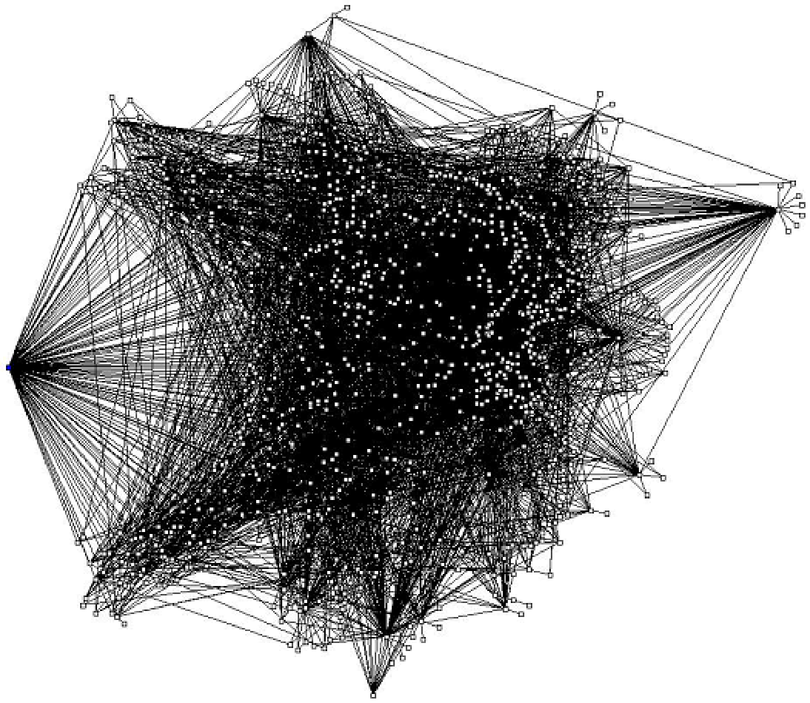


Figure 1.11: Gnutella topology graph captured in 2000 - Source: [M. 2000]

The regular graphs have high pathlength and high clustering coefficients. Random graphs have low path length and low clustering coefficients. As shown in the diagram figure 1.12, regular and random graphs are two extremes of the graph. Starting with a regular graph, with some probability links are changed to connect to a randomly chosen node. The graph shows the two metrics as the probability of changing a link is increased. It can be observed that at probability 0.01, the cluster coefficient is also held high and pathlength also is low. This intermediate topology is known as small world graph [Watts & Strogatz 1998]. Small-world networks have a small diameter and exhibit high clustering. Studies have shown that the Web [Albert & Barabasi 1999, Broder *et al.* 2000], scientific collaboration on research papers [Newman 2001], film actors [Amaral *et al.* 2000], and general social networks [Adamic *et al.* 2003] have small-world properties.

Gnutella's traffic is analysed and found that it has the small-world characteristics. Small-world means short pathlength and high clustering coefficient. Pathlength distribution is shown in figure 1.13. Short path length indicates reduction in number of messages to reach any node. But in Gnutella the query is propagated by flooding the neighbours. This means that number of messages increase exponentially at every hop, although the

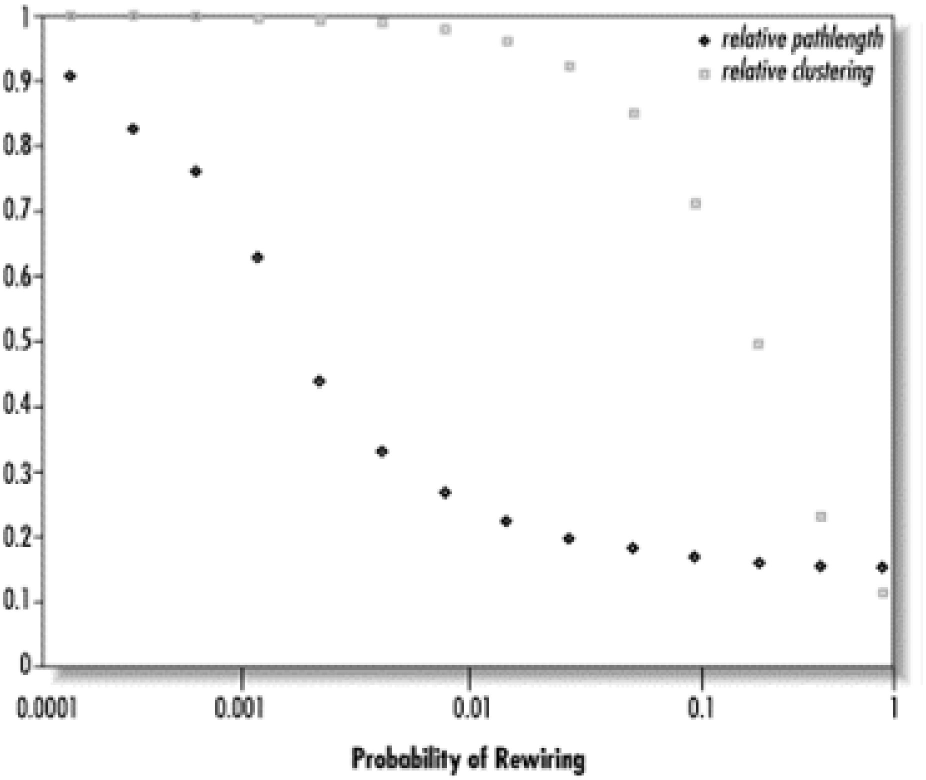


Figure 1.12: Pathlength and clustering coefficient at different degrees of random connectivity - regular graph is at the left most side and the random graph is at the right most side. Source: [Oram 2001]

object may be found in few hops itself. Clustering helps in communities where similar interest nodes are grouped together so that the object requested is found in the neighbourhood itself. Gnutella doesn't have any rules or provisions for clustering based on interests of the nodes.

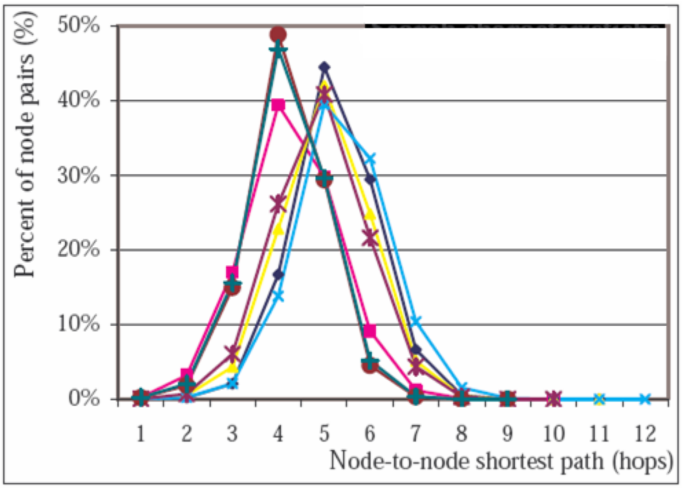


Figure 1.13: Pathlength distribution in Gnutella during March 2001 - Source: [Ripeanu 2001a]

1.9.2.2 Power-law Graph Characteristics

Power law characteristics of Gnutella describe the node degree distribution, while the small world describes characteristics of path length and clustering coefficient as discussed above. The power law distribution is a popular phenomenon in complex networks. Gnutella has been shown to exhibit power-law network characteristics in [Ripeanu 2001b, Adamic *et al.* 2001, Sen & Wang 2002]. Peers connect to a node i with probability $\frac{d_i}{\sum_{j=1}^{N} d_j}$, where N is the set of nodes currently in the network and d_i is the node degree of peer$_i$, which yields a power-law network [Medina *et al.* 2000]. In other words, in power-law networks, the number of nodes with degree k is equal to ck^{-r} where c and r are network constants. In power-law networks, large portion of the nodes have few links and small fraction of the nodes have large number of links. This phenomena is shown to occur in natural networks such molecules, species and also in naturally formed social networks [Broder *et al.* 2000]. The characteristic of such systems is that they are highly stable and resilient to failures. Even if a large number of nodes are removed from the network, still the network continues to function as most of the nodes that are removed from the network have a few links. But a planned attack against highly connected nodes can bring down the whole network [Albert *et al.* 2000]. Figure 1.14 shows the node degree distribution of Gnutella network in May 2001. This graph is linear on log-log scale and gives clear indication of power-law characteristics. But after 4 years i.e. in 2005, study done by [Stutzbach & Rejaie 2005], as depicted in figure 1.15, shows the deviation from power-law characteristics when Gnutella is enhanced with super-peer structure.

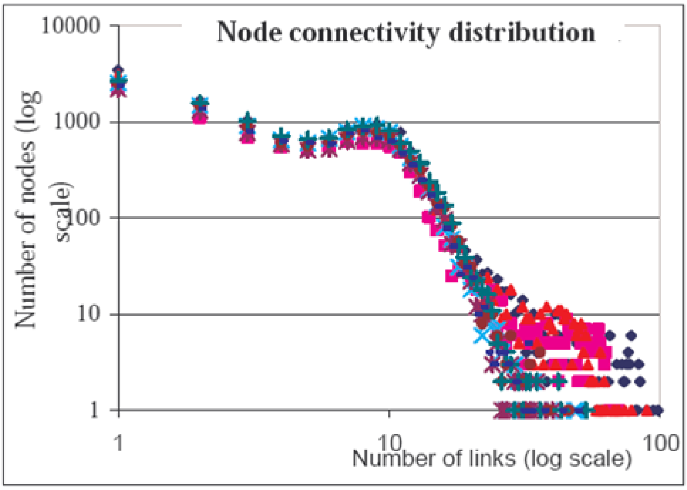


Figure 1.14: Node degree distribution of Gnutella during March 2001 - Source: [Ripeanu 2001a]

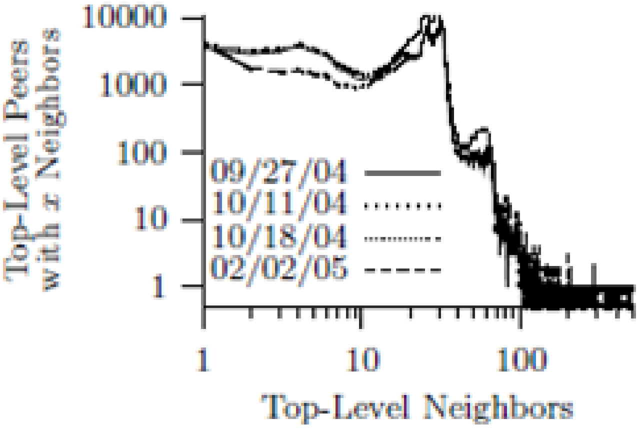


Figure 1.15: Node degree distribution of Gnutella during 2005 - Source: [Stutzbach & Rejaie 2006]

1.9.3 Kazaa and Gnutella 0.6

In pure decentralized systems, search for files is limited to certain hop-limit due to huge number of messages generated by flooding algorithm used. To improve upon that, certain high capacity nodes in terms of computation and network are promoted to 'super-peers'. A certain number of leaf nodes are connected to each super peer. Super peer indexes the files available with its leaf nodes and also maintains connections with other super peers. Any leaf node sending search query first reaches super peer. Super peer looks at its local index and if there is no match it will forward the request to other super peers connected to it. There is flooding among the super peers. Since super peers are few in number compared to total number of nodes, the reachability of the query is greatly enhanced. Since super peers are also prone to churn and failures the leaf nodes are generally connected to more than one super peer. This will increase the robustness of the network. In Gnutella 0.6, a leaf peer is connected to 3 ultrapeers, and each ultrapeer is connected to more than 32 other ultrapeers. Kazaa also follows similar super-node structure. Figure 1.16 shows the search in super-peer networks.

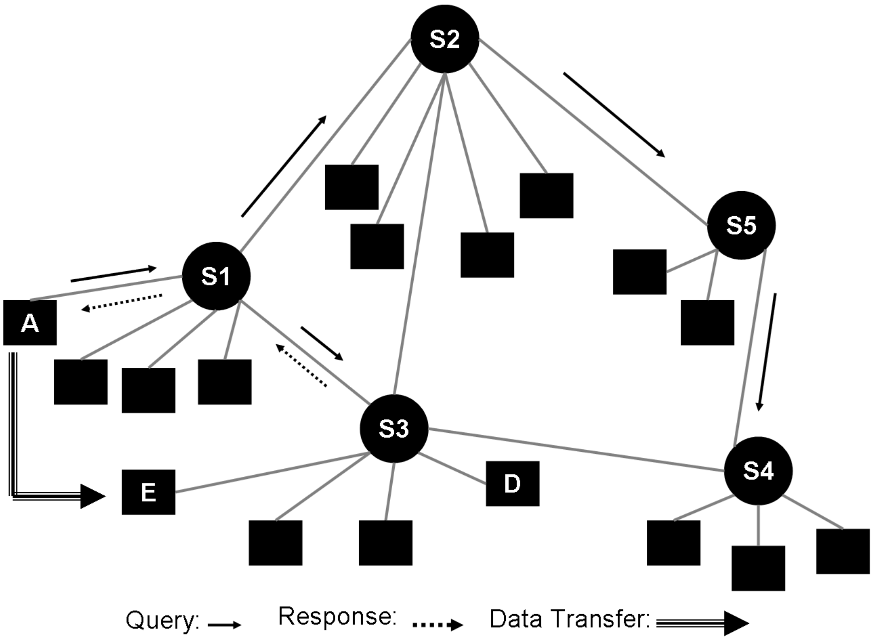
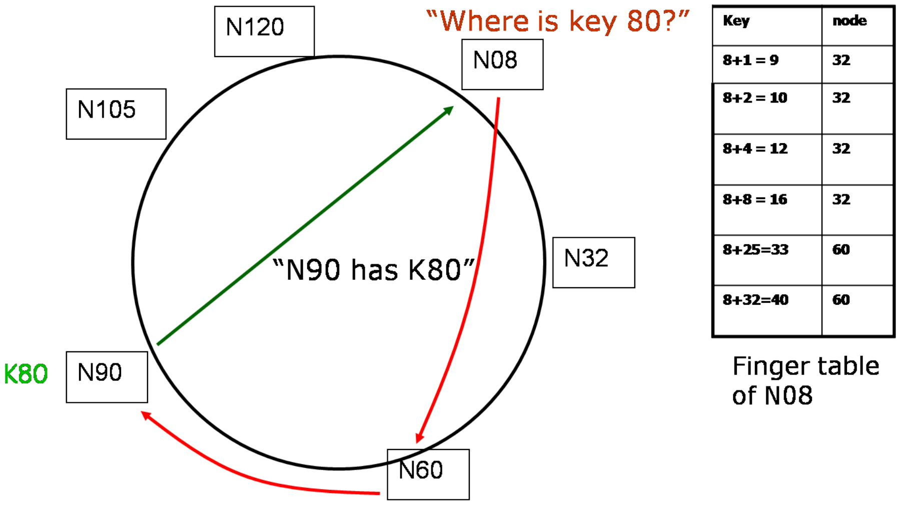


Figure 1.16: Search in super-peer networks - Leaf node *A* queries super-peer *S*1. *S*1 checks with itself and forwards to all neighbour super-peers. Reply is sent by *S*3 since the file is at its leaf node *E*. *A* gets file from *E* by direct connection

1.9.4 Chord

Chord was developed by [Stoica *et al.* 2001] at MIT, USA. It is one of the first peer-to-peer overlay network based on distributed hash table (DHT). Chord uses an m-bit identifier space with range 0 to $2^m - 1$. Chord assigns an identifier using consistent hashing [Karger *et al.* 1997] to a node in this range. Similarly the objects with keys are also assigned identifiers. Consistent hashing ensures with some probability that the keys are distributed among the nodes uniformly. The identifier space is organized in a ring topology as shown in figure 1.17. All network operations are carried out in clock-wise direction on the ring. The object with key K_j is stored by the node N_i immediately following j on the ring. If the object's key coincides with node identifier, that object is stored at that node.

Each node has a reference to its successor node on the ring in the clock-wise direction. To have robustness in case of failures, Chord maintains successor list of size r. Correctness in successor links ensures correctness in lookup. Just the successor list will have worst-case lookup complexity $O(n)$ where n is the overlay size. To improve the efficiency of lookup protocol, each node maintains a finger table of size m. The i^{th} entry in finger table is $successor(n + 2^{i-1})$. Hence finger table maintains links to nodes at $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$ distances from itself. With finger table, in every hop the distance to destination node is reduced approximately by half. This way Chord guarantees lookups in $O(\log n)$ hops.



Key	node
8+1 = 9	32
8+2 = 10	32
8+4 = 12	32
8+8 = 16	32
8+25=33	60
8+32=40	60

Figure 1.17: Lookup procedure in Chord overlay - the id space is in the range 0 to 127. To lookup object with key 80, node 8 contacts its finger 60, whose distance from key 80 is the minimum. Node 60, in turn, does the same.

The lookup request of key k at node i happens in the following way. First node i

checks if k is in $(i, successor(i)]$. If k is found in the range, the request is forwarded to the successor. If not found, it is forwarded to the largest finger in the finger table such that it is $\leq k$. The same procedure is carried out at intermediate nodes. This is depicted in figure 1.17.

Newly arrived node uses consistent hashing to generate its identifier. Then it contacts bootstrapping node, to know its successor node. By running the stabilization protocol, it establishes it's routing table. This protocol is run periodically to learn about the node failures and corrects its routing table. The churn is a problem in Chord as in any structured network. It is proven that if the node's successor list size (r) is $O(\log n)$, lookup can still succeed with high probability even if every node fails with probability of $\frac{1}{2}$.

1.10 Security in Peer-to-Peer Overlays

Security is the fundamental issue to be addressed when the system involves multiple users and their shared resources. Large-scale peer-to-peer overlays involve millions of user identities and their devices contributing to the functioning of the network. Authentication, integrity, confidentiality and non-repudiation are some of the security properties expected to be supported by the system. These issues become significantly more challenging than in the case of traditional domains due to distributive ownership, lack of centralized control and lack of global knowledge in large-scale peer-to-peer overlays. Therefore peer-to-peer overlays face additional security risks when compared to security issues in network applications. Some of such security risks come under large-scale impersonation attacks, using peer-to-peer overlay as a platform for distributed denial of service attacks, and file pollution. Peer-to-peer overlays introduce an additional layer called "overlay layer" which involves specific security risks which are not common in the Internet applications.

1.10.1 Sample Attacks and Threats

Theft is a major risk discovered in the studies of file sharing systems security conducted by [Johnson & Dynes 2007, Johnson *et al.* 2009]. Adversaries took advantage of lack of confidentiality in communication and inadvertent disclosures by innocent users to access confidential information. USA Today article in September 2007 [Johnson 2007] has stated

Table 1.8: Search performance comparison of some overlays - n refers to overlay size. Space complexity: no. of entries in routing table at each node, time complexity:no. of overlay hops per query , and message complexity: no. of messages generated per request

Network Type	**Search Algorithm**	**Space Complexity**	**Time Complexity**	**Message Complexity**
Freenet	Key Based Routing (KBR)	$O(1)$	$O(\log n)$	$O(\log n)$
Gnutella	Flooding	$O(1)$	Varies with individual requests depending on object distribution. It is constant with respect to overlay size as there is timeout limit on search request	$O(d^k)$ where d refers to average node degree and k is TTL
Kazaa, Gnutella 0.6	Flooding	O(1)	Varies with individual requests depending on object distribution. It is constant with respect to overlay size	$O(d^k)$ where d refers to average node degree and k is HTL (hops-to-live). But this is only among the super peers
Napster, Bittorent	Contact central server or tracker	$O(n)$ at central server or tracker	$O(1)$	$O(1)$
Chord	Key based routing using DHT as substrate	$O(\log n)$	$O(\log n)$	$O(\log n)$

that the number of people affected by lack of confidentiality in P2P protocols is in the order of hundreds and the total amount lost is in the order of hundreds of thousands of dollars. The most well-known security violation is illegal copy and distribution of multimedia content and software. [Zeng *et al.* 2006] has stated that annually 447 million dollars are a loss due to online piracy. Peer-to-peer networks open avenues for serving as platforms for distributed denial service of attacks (DDoS). In [Naoumov & Ross 2006], authors proposed that by poisoning indexes and routing tables in the nodes of the peer-to-peer network, target-host is bombarded with requests and thus causing denial of service. Also, peer-to-peer networks offer an attractive platform for spreading viruses by infecting popular downloads with viruses.

1.10.2 Peer-to-Peer Overlay Layer Attacks

Attacks on the overlay can be divided into attacks on message routing, Sybil, and Eclipse attacks.

1.10.2.1 Attacks on Message Routing

There are several known attacks that work by modifying the node's routing tables. Routing tables are key resources for the stability of structured networks. In [Castro *et al.* 2002a], it is noted that an attacker can obtain specific node Ids and strategically position in the overlay such that he controls the access to specific peers or objects. Also poisoning of routing tables and message forwarding attacks are possible. These attacks are discussed in [Castro *et al.* 2002a] and [Wallach 2003].

1.10.2.2 Sybil Attacks

In this attack, an entity can represent itself as multiple identities in the overlay and thus gain control over disproportionate resources. This attack was first pointed out by [Douceur 2002]. In this attack, an attacker can influence the reputation of the systems and objects and also carry out malicious attacks like disrupting the overlay operations. In this thesis, the focus is on developing novel distributed approaches to limiting Sybil attacks. In Figure 1.18, it can be observed that there are entities and identities (blue circles). Some entities forge themselves as multiple identities in the network. Because of their large fraction of identities, the entities can control the network. To differentiate between a real identity and a sybil identity is very difficult. It is stated in [Douceur 2002] that without a centralized authority it is not possible to completely eliminate the sybil identities from the network. Since peer-to-peer overlays fit decentralized mechanisms, in this thesis, we have put effort to develop algorithms to limit sybils in a distributed way.

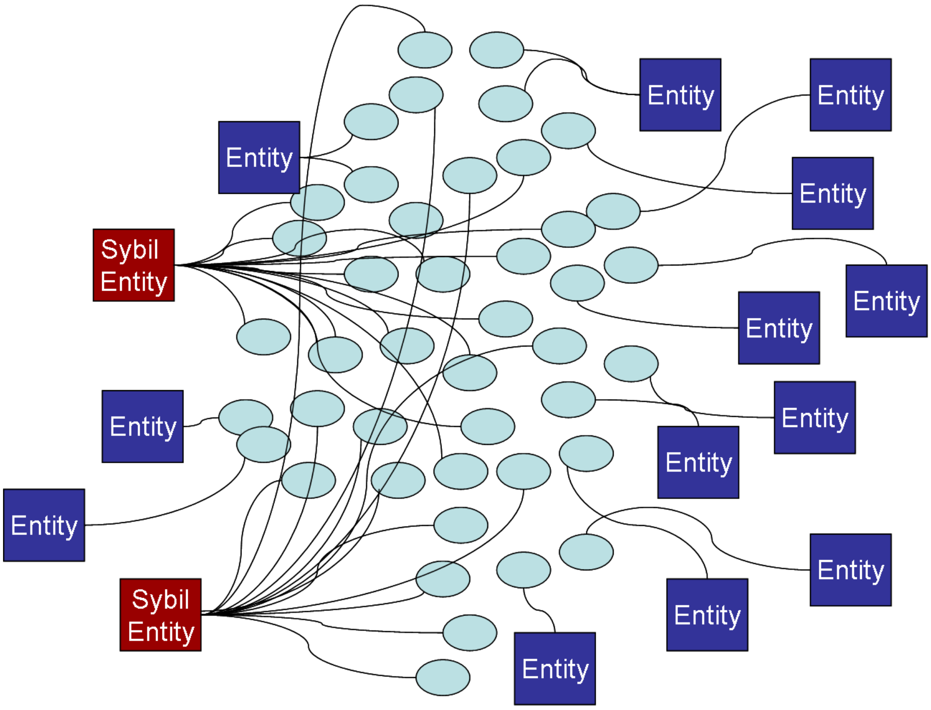


Figure 1.18: Sybil entities and identities in a peer-to-peer network - Entity is represented by a rectangle and identity by an ellipse

1.10.2.3 Eclipse Attacks

In these attacks, one first gains control over a large number of nodes along strategic routing paths and then separate the network into different sub-networks. Traffic between the sub networks has to go through one of the attacker's node. This way this attack can be used to disrupt the network in a systematic way or propagating false files in a systematic and fast-paced way.

1.11 Scope of the Thesis

Peer-to-peer overlays have many research issues to be addressed such as efficient search in unstructured networks, complex queries in structured networks, overlay adaptation to churn, user privacy and trust, securing overlays from different types of attacks using decentralized approaches etc. In this thesis, the focus is on two issues: (i) developing distributed approaches to improve search performance in unstructured networks and (ii) developing distributed approaches to limit Sybil attacks in large-scale peer-to-peer overlays. P2P networks are large in size and the node and object placement is random in unstructured networks. Providing a efficient search for finding objects in such overlays faces many challenges. Sybil attacks are hard to detect even using centralized algorithms. Here we propose distributed mechanisms.

The thesis is organized as follows. Chapter 2 discusses the previous approaches in the literature to these two issues. Chapter 3 discusses the proposed approaches to improve search performance, chapter 4 discusses a QoS model for P2P file sharing networks and Chapter 5 discusses the proposed approaches for limiting Sybil attacks. Chapter 6 gives the conclusions.

Chapter 2

Problem Definition and Known Approaches

Peer-to-peer overlay networks have applications in variety of domains, namely communication, file-sharing, multi-media distribution, file-storage, collaboration, and QoS routing in the Internet, etc. In this thesis, the focus is on file-sharing peer-to-peer systems. Within this domain, there are fundamental issues to be addressed by researchers. Some of them are search, security, churn, load-balancing, topology adaptation with overlay dynamics, etc. In this thesis, the focus is on improving search in unstructured overlays in terms of efficiency and quality of service and developing distributed approaches to limit the Sybil identities in peer-to-peer overlays. Search scalability in unstructured overlays, at the same time giving quality of service is one of the rigorous research areas as most of the well-known peer-to-peer file sharing systems are built on unstructured overlays. In this chapter, the study of approaches proposed in the literature is presented. [Douceur 2002] has proved that Sybil attack can be fully solved only by a central authority which issues certificates based on some physical evidence such as photo identity etc. But this will pose as a big barrier for their deployment and limit its reach to users. Therefore, distributed approaches become a necessity to overcome these pitfalls. File-sharing systems involve rating of objects by their users. Such reputation schemes can be thoroughly influenced by Sybil entities. In this chapter, approaches proposed in literature to limit Sybil attacks are also presented.

2.1 Search in File-sharing Overlays

The amount of data in the web was estimated to be 66.8 to 91.9 Petabytes [pro 2003]. The amount of data stored by Google was estimated to be 24 Petabytes of storage. The size of data shared by Kazaa file-sharing system, as of 15th October 2005 was estimated to be 54 Petabytes [sta 2005]. This huge data can be accessed by all users of the network only if search is effective. Thus search is of vital importance in file sharing systems without which most of the data will remain hidden and inaccessible to users.

2.1.1 Search Problem

The search problem in a peer-to-peer overlay refers to finding any given data item in a scalable manner. More specifically, given a data item stored at some dynamic set of nodes in the overlay, we need to locate it [Balakrishnan *et al.* 2003].

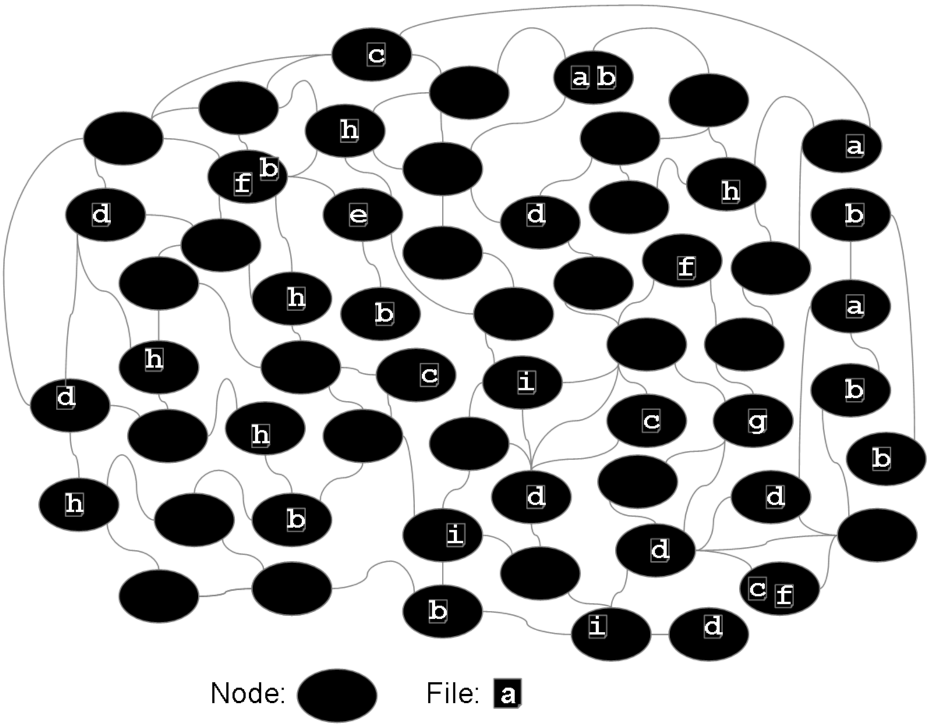


Figure 2.1: File sharing in a P2P overlay - this is an unstructured overlay. File are shared by nodes as per their wish. Some nodes don't share any files

In an unstructured overlay, nodes share files as per their wish. As shown in figure 2.1, files are present at random locations. A file is shared by many nodes or just one. Some nodes share many files and some not even one. No node in the overlay has the global knowledge of the entire topology. A node knows only few links connecting it to the rest of the overlay. Nodes are transient, i.e. any node can leave the system at any time. This type of situation is called as 'churn'. Churn can make the network partitioned.

In such a system, the only way a node can search for a file is through its neighbours. Such a search can result in querying every node on the overlay. This is highly undesirable as it consumes resources heavily and results in burdening other nodes. So the search problem is about locating the files in a dynamic overlay in a scalable way and at the same time maintaining search quality.

2.1.2 Factors Affecting Search Performance

Search in an unstructured peer-to-peer file sharing system is dependent on many factors centred on overlay topology, data placement and routing. Search in a decentralized system involves many other nodes in the overlay. How many nodes and how much they are involved depends on the overlay organization and it's routing. Pathlength and clustering of nodes in the overlay affect the efficiency and quality of search results. Pathlength is the average number of hops required to reach any node from any node. If the links between the nodes involve long-range links i.e., links which connect to the nodes beyond the local neighbourhood, then pathlength is reduced. Pathlength has effect on bandwidth and processing overhead. Smaller pathlengths provide quick coverage of the network in smaller hps and thus reducing the message traffic. Some researchers have devised algorithms to adapt the topology dynamically to keep the pathlength as low as possible. Clustering of the nodes in the local neighbourhood has a direct effect on the quality of results. If the clustering is based on the semantics of files then it is most likely that relevant files will be available in the local neighbourhood itself. Several researchers have proposed building overlays centred on the nature of their interests, content etc. Node heterogeneity in terms of network capacity and processing capacity is inherent in any large scale networks. Search and maintenance functions of the overlay are distributed in proportion to node's capacity.

In unstructured decentralized networks, routing involves decrementing Time-to-Live (TTL) at every hop. Searches with larger TTL generate enormous traffic consuming huge network bandwidth and generating duplicate queries. Several search algorithms proposed in the literature that control TTL limit iteratively, adaptively etc. Routing also involves choosing the nodes to forward the query. Number of nodes are chosen at every hop has influence on the query traffic generated. Choosing the nodes can be a blind decision or an intelligent decision. Intelligent decisions contribute to quality of results. Intelligence is derived from the history maintained at the nodes. Building indexes helps in searching, increasing both efficiency and quality of results. But maintenance of indexes in a dynamic population of nodes and objects adds to the overhead in terms of bandwidth and processing overhead on nodes. BloomFilters[Bloom 1970] help in reducing the size of the messages thus reducing the overhead. Replication of files or keeping several copies of the same file on different nodes across the overlay enables load balancing, availability and also efficient and quality of search.

In the next section, the approaches proposed in the literature are categorized and presented with respect to these parameters.

2.2 Approaches to Improve Search

A good search mechanism is one which allows users to effectively locate desired data in a resource-efficient manner [Daswani *et al.* 2002]. In unstructured overlays, there are several challenges like their large size, transitive population of nodes, heterogeneity, user autonomy etc. There are number of search algorithms proposed to meet these challenges.

The first file sharing network [nap 1999] maintained a centralized index of all the files. Search is carried out by the central server itself without involving other nodes in the network. Central servers have the following limitations.

- Central servers pose as single point of failure. If they are down, the whole network becomes unavailable.
- Can't scale well with increased demand. The processing capacity and bandwidth are throttled.
- Central servers are prone to malicious attacks from envious people.
- Content becomes subjected to censorship-laws of different countries.

With litigations in Napster, Gnutella [gnu 2000] became popular. Gnutella uses a fully decentralized search mechanism which is known as flooding or breadth-first-search [Yang & Garcia-Molina 2002]. In flooding, the query is forwarded to every neighbour node with a certain TTL limit. Advantage of flooding is that initial results are reported very fast. The disadvantage is that the traffic generated increases exponentially at every hop. Every node in the overlay handles the query traffic that is proportional to the total number of nodes in the overlay [Adamic *et al.* 2001]. [Lv *et al.* 2002a] notes several limitations of flooding, as below:

- Heterogeneity is not taken into account. The low capacity nodes are burdened with the same load as the higher capacity nodes.
- Choosing an appropriate TTL is not easy. If the TTL is too high, the node unnecessarily burdens the network. If it is too low, the node might not find the object even though copy exists somewhere.
- Duplicate queries are those which are sent to a node by its neighbours, although the node has received that same query and processed it. Such messages are pure overhead. Although Gnutella has duplicate detection mechanism that avoids processing them, but that doesn't prevent the duplicate messages being sent to the neighbours. The figure 2.2 shows the duplicate messages in the network

[Lv *et al.* 2002a] proposes a method for addressing the TTL selection problem which they call it as 'expanding ring' method. Requester node searches with increasing TTLs until a success is found. The results show that the method reduces the number of messages at the cost of delay in getting the response. [Yang & Garcia-Molina 2002] proposed a flooding policy known as iterative deepening which performs multiple breadth-first searches with successively larger depths. Suppose the policy $P = \{a.b.c\}$ is defined at a global level. The first search is carried out with TTL as a and second search with TTL as b and the third search is carried out with TTL as c . The advantage with this is that the requesting node has the freedom to decide whether to go for flooding at higher TTL levels or not. The disadvantage is that the response can be huge in case the objects are not found in lower TTLs. Also it generates a lot of duplicate messages. This is because

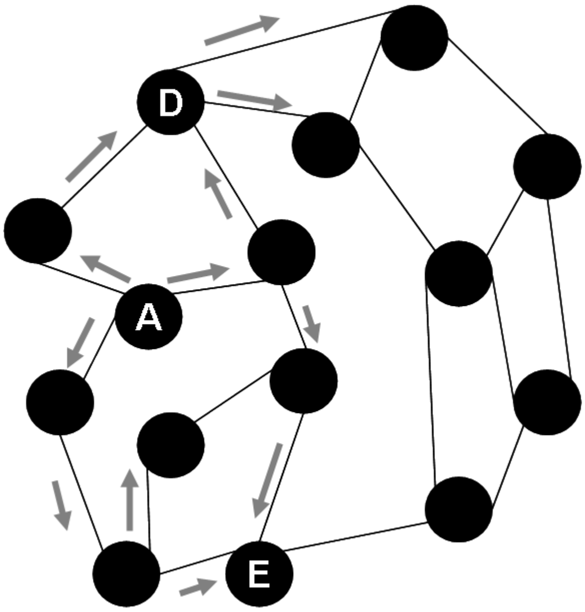


Figure 2.2: Flooding in P2P overlay with TTL=3 - duplicate messages can be observed at nodes *D* and *E*

the query with next level TTL has to go through all the nodes which would have already received this query although they don't process it.

The use of random walk method to find an object is proposed by [Lv *et al.* 2002a]. In this method, the query is forwarded to a randomly chosen neighbour at each hop until the object is found. k-random walkers can be used to reduce the delay. k-walkers after T steps reach roughly the same number of nodes as 1-walker after kT steps. This way by using k-walkers, the delay is reduced by a factor of k. 1-randomwalk is depicted in figure 2.3. 2-randomwalk is depicted in figure 2.4. The termination of random walkers is caused either by TTL or checking with the source node or by maintaining status of the queries received at each node.

The key difference between k-random walk and flooding is that, the granularity of coverage is small in random walk. An additional step of search results in k messages in random walk where as in flooding it results in d^{TTL} number of messages, where d is the average degree of the node.

The basic flooding and random walk searches have limitations. Flooding is inefficient and doesn't scale and random walks have long response times. These basic methods are improved upon by several ways like overlay topology, routing criteria, object placement etc. which are discussed below:

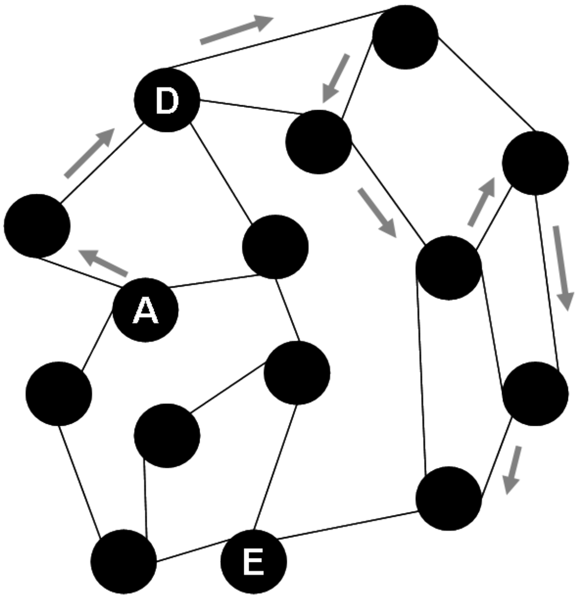


Figure 2.3: Randomwalk in P2P overlay - randomwalk is initiated by node A. At each hop the query is forwarded to a randomly selected neighbour. The randomwalk is terminated when TTL reaches 0.

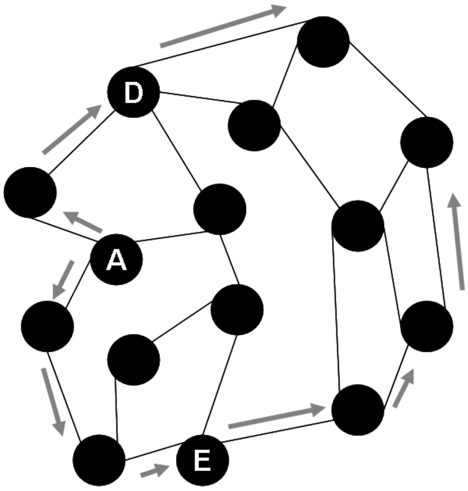


Figure 2.4: 2-Randomwalk in P2P overlay - two walkers are initiated by node A. At each hop the walker is directed to a randomly selected neighbour. The walker is terminated when TTL reaches 0.

2.2.1 Adapting Topology

[Pandurangan 2001] proposes a scheme to build a connected overlay network of constant degree and logarithmic diameter based on simple rules of choosing neighbours to connect to from a cache maintained by a host which is always available. It does so with no global knowledge of all the nodes and their topology in the network.

[Ratnasamy *et al.* 2002] proposes a 'binning' strategy to build low-latency networks. The nodes when they join measure their distance with respect to a set of land mark machines in the Internet. Based on either the order of distances or the level of absolute distances, the node chooses the bin to put itself in. This binning also helps in choosing the peer with minimum latency to download the content from. This approach works without any global knowledge and it is scalable.

[Lv *et al.* 2002b] proposes a simple capacity based topology adaptation mechanism based on the neighbour statistics. In this method, if a node finds that node is sending queries at a rate which it can't handle, node finds another node among its neighbours that has the maximum spare capacity and directs the qury to that node. By adapting links in this way, the nodes are directed towards higher capacity nodes.

GES [Zhu & Hu 2006] uses information retrieval algorithms such as Vector Space Model (VSM) and relevance ranking algorithms to improve search performance in Gnutella. GES uses a distributed topology adaptation algorithm to organize semantically relevant nodes into same semantic groups by using the notion of node vector. Given a query, GES employs an efficient search protocol to direct the query to the most relevant semantic groups for answers, thereby achieving high recall with probing only a small fraction of nodes.

There are several other works that focus on topology adaptation to improve search. Some of them are described in [Condie *et al.* 2004], [Xiao *et al.* 2005], and [Wang & Wang 2006].

2.2.2 Replicating Objects

Some schemes have used replicating objects in the network in order to increase efficiency and quality of results. Major considerations in replication are selection of objects and selection of sites. There are various replication techniques explored in literature like path replication, square root replication [Lv *et al.* 2002a], Pull-Then-Push replication [Leontiadis *et al.* 2006] etc.

2.2.3 Modifying Routing

There are several algorithms proposed in literature which explore possibilities of increasing search performance by modifying routing mechanism of the search queries.

2.2.3.1 Neighbour Selection

In flooding, the query is forwarded to the entire set of neighbours blindly. Instead of making the selection blind, it can be done with intelligence. In directed breadth-first-search [Yang & Garcia-Molina 2002], the queries are forwarded to selected neighbours. The selection of neighbours is based on a history maintained for each neighbour with the characteristics such as number of query results returned and network latency of that neighbour etc. Here, this selection is only done by the requesting node and the forwarding nodes do flooding. In intelligent search [Kalogeraki *et al.* 2002], the selection of neighbours is done by similarity metric that indicates how well the neighbour has answered similar queries in the past. The similarity is measured by cosine similarity model. Unlike directed BFS, in this case every forwarding node selects a subset of neighbours and forwards the query. Intelligent search works better in locality based overlays.

In Adaptive Probabilistic Search (APS) [Tsoumakos & Roussopoulos 2003], *k*-randomwalk is modified by incorporating probabilistic selection of neighbours to forward at each hop. Each node maintains probability values for each of the neighbours. The probability value is computed based on the results returned for the past queries. The node starts with a guess value initially. As the queries are triggered in the network, the guess values are adapted to reflect the neighbour's response to the queries. While forwarding the query, the neighbour with the maximum probability value is selected. Compared to *k*-randomwalk, this mechanism has the same search efficiency but there is an increase in the quality of the search results. [Zhang *et al.* 2007] considers object popularity and neighbour node degree in combination with probability to select the neighbours which has further increased the efficiency of search. [Zhuge *et al.* 2005] developed a neighbour ranking method based on trust metric. [Yuan & Yin 2007] proposed a neighbour ranking method that considers number of shared files, number of query hits returned by the neighbour, and the link capacity.

[Adamic *et al.* 2001] has shown that in power-law graphs, if random walk is targeted to a well connected node instead of random selection, the search time and search coverage is greatly improved. In a scalable overlay "Gia" proposed by [Chawathe *et al.* 2003], the search is carried out using random walks that are targeted to nodes with high capacity. In Gia, neighbours are free to say that if they are ready to accept queries or not. Nodes distribute tokens to neighbors depending on their query handling capacity. A node selects the highest capacity neighbor for which it has got the token which helps in flow-control.

2.2.3.2 Adaptive TTL Selection

For both flooding and random walk based searches, the TTL plays an important role of determining when to terminate the search. Therefore in [Lv *et al.* 2002a], it is outlined that TTL should be an adaptive parameter rather than a static value. In [Fisk 2005] a dynamic query routing protocol is proposed. In this protocol, the source peer first sends query packets towards a few neighbours with a small TTL value. The purpose of this probe phase is to have an initial estimate of the popularity of the searched item. Then an iterative process takes place. In each iteration, the source peer estimates the number of peers to be contacted in order to obtain the desired number of results; then it calculates the TTL of the query packet to be sent to the next neighbour. This iterative process stops when the desired number of results is returned, or all neighbours have been visited. [Jiang & Jin 2005] has enhanced the dynamic query routing by using greedy method of selecting only one neighbour in each iteration. [Bisnik & Abouzeid 2007] have derived an expression that gives the relationship between popularity of the object, number of random walkers, and the TTL value. They have suggested that if the object popularity is known, then the expression will give the number of random walkers and the TTL value so that the search performance is optimized. [Thampi & Sekaran 2009] has categorized the network into peers and power peers. Power peers have the capability of enhancing the TTL value if they find that the object is very rare in the network.

2.2.3.3 Routing using Indexes

Indexing is the most important tool for searching [Baeza-Yates & Ribeiro-Neto 1999]. Index building is about creating and maintaining data structures that have files and their location information.

In local indices technique [Yang & Garcia-Molina 2002], each node maintains an index of data of all nodes within *r* hops of itself. When a node receives the query, it processes it on behalf of all the nodes within *r* hops of radius. Creating and maintaining such index involves extra overhead on the network. In routing indices technique [Crespo & Garcia-Molina 2002], index is created for different topics in different routes. The index is used for choosing a neighbour to forward the query. In this technique, the aggregate updates are exchanged among the nodes to keep the index up to date. By sending only the aggregated vectors, the overhead is reduced. In attenuated bloom filter technique [Rhea & Kubiatowicz 2002], an index for every neighbour and up to *d* number of hops is maintained. Index for neighbour *n* stores the bits of objects generated by Bloom Filter [Bloom 1970] available at hop *d* through neighbour *n*. The index is maintained by exchanging the changed bits. In [Zhang & Hu 2007], a global but partial index is built using a Distributed Hash Tables (DHT) on top of the unstructured overlay. The index consists of peer's top interests which is reflected by the nature of queries and objects a peer has. Changes in the interests of peers involve changing the neighbours. In [Kumar *et al.* 2006], super peers develop multi-attribute based content index by observing the QueryHit messages that pass through it. They use this index to efficiently forward the queries to selected neighbouring super peers. There is a separate index table for every attribute. The entries are deleted based on an age factor. In advertisement based technique [Gu *et al.* 2007], a local index is built by requesting neighbours up to certain hop level to send their ads or indexes. In eSearch [Tang & Dwarkadas 2004b], index for every term is created. One node is responsible for maintaining an index of one term. Nodes analyse their documents and find top terms and publish them to the respective nodes responsible for those terms.

2.2.4 Search Issues Addressed in this Thesis

There are several dimensions over which search problem can be investigated. One of the dimensions that is considered in this research work is that adapting search function to user needs. There is variety of search needs in a file-sharing overlay. Search can be for a file to be downloaded, a music file to be streamed etc. Adapting search to provide for the diverse search needs of users is one of the issues addressed in this research. Second issue is that enhancing search efficiency by building an index without sending any extra messages. Third issue addressed is that enhancing the search efficiency and quality by making probabilistic decisions at every hop with the help of a soft computing technique.

2.3 Study of Sybil Attack in File-sharing Overlays

[Douceur 2002] has first given the name 'Sybil' to the type of attacks where there can be many to one correspondence between identity and entity in peer-to-peer networks.

2.3.1 Defining Sybil Attack

Sybil attack is an attack where an entity in a peer-to-peer network can masquerade itself as multiple simultaneous identities in the network [Douceur 2002]. A peer-to-peer overlay file sharing network consists of set E of infrastructural entities e. An identity is an abstract representation that persists across multiple communication events. Each entity e attempts to present an identity i to other entities in the system. Each correct entity e will attempt to present one legitimate identity. Each faulty entity f may attempt to present a legitimate identity and one or more counterfeit identities. Ideally, the system should accept all legitimate identities but no counterfeit entities.

The problem with such duplicitous mapping of many virtual identities on to one entity is the collective influence a single user can exert on the decisions & working of the entire network if the multiple identities created by the user form a significant fraction of the peer-to-peer network. This problem is pervasive in all distributed systems.

This attack is possible in any distributed network. But peer-to-peer network is an attractive field for this attack due to its lack of central control, and large size. Peer-to-peer networks have huge resources like processing power, bandwidth and storage contributed by the participants. These resources pose attractive target for attackers as the attackers can exploit these resources for selfish purposes. Although this kind of attack is possible in mobile ad-hoc networks and sensor networks, but there the nodes are constrained by their physical characteristics [Dinger & Hartenstein 2006]. Since peer-to-peer network is built at application layer the physical constraints don't limit the Sybil attacks.

It is not very difficult to set up a Sybil attack. Creating an identity in the network is as simple as starting another instance of the peer-to-peer client. If a malicious user has a vast pool of resources at his disposal, he can create a large number of such identities.

2.3.2 Observed Instances of the Sybil Attack

Due to large size of P2P networks it becomes difficult to assess trustworthiness of a peer with whom we interact. Therefore reputation systems are used to aggregate the collective experiences of peers about other peers [Resnick *et al.* 2000]. When a peer needs to interact with another with whom it has not interacted so far, the reputation system helps in making opinion about that peer. In online systems like Amazon, eBay etc reputation systems are used to aggregate the ratings of sellers and goods. Such ratings have impact on business transactions [Depken & Gregorius 2008]. In file-sharing overlays, reputation of files affect users opinion whether to view that file or not. Such benefits attract malicious attempts to manipulate the reputation systems. [Gyongyi & Garcia-Molina 2005] have reported that the web page rankings can be manipulated by setting up a link farm. [Bhattacharjee & Goel 2005] [Cheng & Friedman 2005] have reported similar instances of manipulations using Sybil attacks. Sybil attack is commonly used to fool Google's PageRank algorithm [Bianchini *et al.* 2005]. PageRank algorithm is one of the most commonly used algorithms to compute the reputation of peers in reputation systems [Kamvar *et al.* 2003].

The major problem in peer-to-peer computational systems such as SETI@home is that, server should ensure that the clients are not cheating by submitting deceptive results without fully performing all the computations specified. One way to detect this cheating is to allocate the same task to multiple clients. But this redundancy can be subverted if there is an agreement among the clients that they would return the same manipulated-result. In internet it is possible that all these clients can be instances of the same devious entity who can synchronize the outputs of all the clients and thus mislead the server [Yurkewych *et al.* 2005].

Sybil attacks create false routes in mobile ad hoc networks [Hu *et al.* 2002]. Sybil attacks can disturb anonymous systems such as Tor by revealing user identities of anonymous routing protocols [Dingledine *et al.* 2004]. Pastiche is a file storage system built on Pastry overlay. Sybil attacks can subvert the distributed quotas by free-riding cooperative file storage systems [Cox *et al.* 2002].

2.3.3 Sybil Attack Vs Collusion

Collusion attacks and Sybil attacks appear the same in their purpose but they do differ in the means [Cheng & Friedman 2006]. Collusion is a strategy where nodes mutually agree to subvert a network policy in a cooperative way. In Sybil attack, a single entity creates multiple identities which are then used by the entity to subvert a network policy. In collusion, every colluding node wants to gain advantage at the expense of its participation in the subversion. But in Sybil attacks, all identities are controlled by one entity and therefore there is no competition for gaining advantage. In collusion attack, the number of nodes are not an large as in Sybil attacks. In Sybil attack creating large number of identities is easy and cheap.

2.3.4 Characteristics of Sybil Attacks

The following are the some of the characteristics of Sybil attacks:

- The strength of Sybil attack depends on the number of identities it creates and how much fraction of the network they occupy. The influence on the network is exerted as a group but not at the individual identity itself. Therefore it is difficult to resolve a Sybil attack at the identity level.

- Having a centralized server issuing the logins or providing authentication itself is not sufficient to prevent Sybil attacks. Amazon, eBay etc have centralized authentication systems but still they face Sybil attacks. The principle behind preventing Sybil attack is that one should be able to map the real infrastructure entity to the virtual identities and then put a limit on such number of mappings. Such a system requires one to authenticate the identity by a physical proof such as photo identity card, credit card etc. Such a restriction on enrolling new entities into the system severely limits the spread of systems among users.

- Sybil entities need not necessarily be creating disturbance in the network such as launching distributed DoS attacks, or dropping the packets or poisoning the routing tables, partitioning the network etc. It may be difficult to say what a Sybil identity is doing is wrong. It may be doing the same thing like any other honest identities. For example, a Sybil entity can position its identities strategically at different places in the network and make sure that every packet in the network will pass through at least one of its identity so that Sybil has control over the network. Looking at the identity, one can't say it is doing something malicious. It should be determined at the network level. Another example is that a Sybil entity can increase the reputation of a particular file by making all its identities respond positively to the reputation metric. Looking at the individual identity it is difficult to say that what it is doing is wrong because every identity has free will to respond positively or negatively to the file.

2.4 Approaches to Limit Sybil Attacks

[Douceur 2002] proved that it is not possible to completely eliminate the Sybils in a peer-to-peer network without a centralized authority which can verify the one-to-one correspondence between identities and entities. He described puzzle methods that exploit communication, storage or computational resource constraints. He proved that computational puzzle methods are not viable. In these puzzles, the verifier sends a large random value to every other identity it wants to verify. These identities must then compute the solution within a constrained amount of time. If an entity has more than one identity it will fail to compute the solution within this time. The paper says that this can be circumvented by taking help of other powerful nodes. Thus he advocates the existence of a central authority to prevent Sybil attacks.

Solutions to Sybil attack can be categorized as challenge-response imposing constraints on resources, binding the identity to physical characteristics, central authority certification, characteristics of social networks based on trusted connections, based on Sybil behavioral characteristics and incentives.

2.4.1 Challenge-Response

The goal of resource testing is to attempt to determine if a number of identities possess fewer resources than would be expected if they were independent. Challenge-response utilizes puzzle methods that exploit communication, storage or computational resource constraints of the participating nodes. In these puzzles, the verifier sends a large random value to every other identity it wants to verify. These identities must then compute the solution within a constrained amount of time. If an entity has more than one identity it will fail to compute the solution within this time. These tests include checks for computing ability, storage ability, and network bandwidth, as well as limited IP addresses [Levine & Margolin 2006]. [Douceur 2002] says that this can be circumvented by taking help of other powerful nodes and therefore, advocates the existence of a central authority to prevent Sybil attacks.

[Borisov 2006] proposes to use computational puzzles to defend Chord from Sybil attacks. In Chord, every node sends periodic ping messages to its neighbors. This scheme proposes that along with every ping message, a sequence number and a challenge will be sent to neighbor y_i. The challenge is actually transformation of all neighbor identities, latest sequence numbers and the latest challenges received from them. The SHA1 hash of concatenating together the neighbor y_i identity, and sequences number and challenge and a random number and the challenge generated by the node in the previous round forms the challenge for the next round. The puzzles are formed out of these challenges and sequence numbers. The dependence of puzzle on the values received from the neighbors makes it verifiable by the neighbors also. The author proposes that every node needs to solve the puzzle by the time next ping message received. Even honest nodes are expected to solve these puzzles. Here the heterogeneity of the nodes in their computation capacity is not addressed.

[Rowaihy *et al.* 2007] present a hierarchical admission control system where at every level computation puzzle are used to validate the identity. The system creates a tree where the root must be trusted and reliable. The root allows other trusted systems such as major ISPs to join the system. These in turn allow smaller providers which in turn allow ordinary users to join the system. When an ordinary wants to join the network, it contacts one of the leaf nodes y of the hierarchical system. y gives a puzzle of guessing R to x. The puzzle is $hash(BK_x \| TS \| R)$ where TS is a timestamp, BK_x is the x's public key, R is the random number generated by y. The x must solve the puzzle by guessing R so that it matches the value specified by y. Once x solves the puzzle correctly, y issues a token that is encrypted by the secret key shared by y with y's parent. When x approaches the parent with the token, parent gives another puzzle to x. This way when x solves the puzzle given by the root, then the root gives a special token suing which x can join the network. This scheme only slows down the identity generation but not prevent the Sybil attack. In case of the sybil being part of the hierarchy, the scheme proposes that parents can observe the rate of generating tokens of their children. If it exceeds certain threshold, that node can be removed from the hierarchy. There is also expiration time associated with token so that even if a node tries to acquire large number of identities, the tokens can't be accumulated for long time. Here the honest nodes are also subjected to the same tests.

2.4.2 Binding Identity to Network Metrics

[Bazzi & Konjevod 2005] proposed that an identity can be mapped to its physical location. There are two types of nodes: applicants and beacon nodes. Geometric certificate contains the distances measured between the node and the beacon and signed by both. This approach introduces a equivalence relation where all nodes in one relation can't be distinguished from others in the same relation. Here the defect is that if the Sybil is controlling entities in different relations then it is not possible to detect it. Also the algorithms to measure distance don't give stable values and requires considerable effort to achieve stable values.[Bazzi & Konjevod 2005] proposes a secure distance vector routing protocol that tolerates Sybil attack. They hop-chains using which the destination node can certify remotely its distance to the nodes in the network. This protocol is on the lines of the method proposed by [Bazzi & Konjevod 2005] replacing round-trip delays with hop-counts.

[Wang *et al.* 2005] proposed a concept of net-print. The net-print of a node is built using node's default router IP address, its MAC address and a vector of RTT measurements from the node to designated land marks. Here the node identity is bound to physical network characteristics. This data can be verified by other nodes making the identity theft difficult. If there are several identities operating from the same subnet, they can be challenged with computational puzzles concurrently. The changing network conditions will affect metrics like RTT measurements. So to depend on these measurements to validate an identity requires certain tolerance range on the measured values. This challenges the strict security requirement in values. This approach fails when the node changes its physical location. So this solution doesn't apply to mobile hosts.

[Dinger & Hartenstein 2006] proposed a distributed registration mechanism for Chord. Each identity calculates its id as a hash of its IP address and port number and registers itself at r registration nodes in the Chord ring. These r registration nodes are discovered using hash of IP and an integer j ($1 \leq j \leq r$). Registration nodes maintain list of identities registered with them. If the registration node finds that the number of identities registered with an IP exceed system-wide constant a, then it will reject the new identity. The new node will be accepted only if $\lceil \frac{r}{2} \rceil$ registration nodes confirm the acceptance. This solution will work only if majority of the nodes are honest. Here the mapping is between identity and its IP address. The cardinality of this mapping is controlled through a distributed registration process. In this solution the influence of Sybil is limited to the number of IP addresses it can possess.

2.4.3 Central Authority Certified Node Identities

[Castro *et al.* 2002a] argue that the only practical solution to prevent Sybil identities in the peer-to-peer overlay network is to produce signed certificates that bind node identity to a public key and the IP address of the node. To a prevent a malicious entity from obtaining large number of certificates, one of the ideas authors propose is that each certificate can be issued against a charge. Another is that certificates can bind to real-world identities.

They allow multiple node IDs per IP address. Certainly this solution prevents Sybils but it also slows down the propagation of the network services to new users. For IP based schemes, for the nodes behind NAT-based firewalls, special provisions have to be made.

2.4.4 Based on Social Network Characteristics

[Danezis *et al.* 2005] present a modified Distributed Hash Table (DHT) routing model using a bootstrap tree for Chord network to resist the impact of Sybil attacks. The bootstrap tree is an initial overlay that connects designated attachment nodes that can be used by others to join the overlay. It is assumed that bad nodes or Sybil nodes are connected to the rest of the bootstrap network through a single good node. Chord protocol is modified that a node not only stores the ids of predecessors, successors and fingers but also the path from itself to the nodes it knows. When a lookup query is received, the node distributes the queries among the nodes it knows so that not too much trust is put on one single node. At every lookup step in the iterative model, the node returns all its neighbors and bootstrap paths. A node is trusted if it is on the bootstrap path from the requesting node to itself. The core of Sybil defense mechanism consists of distributing queries around the network in such a way that no small set of nodes is predominantly present on the paths of the queries. It is not mentioned to what extent the logarithmic lookup times can be maintained with this approach. It increases the overhead in lookups.

In Sybilguard [Yu *et al.* 2006], the authors have proposed a distributed algorithm to limit the entry of Sybil identities into a social network, exploiting the fact that there are very few trust edges between an honest and a Sybil group in a social network. They have designed a protocol in which the verification of a new entry into the network is done by intersection of random routes. The problem with these approaches is that they work only with networks that have evolved based on social trust relationships. This is not the case in a majority of the existing public peer-to-peer file sharing systems such as Gnutella, Freenet etc.

[Lesniewski-Laas & Kaashoek 2010] present Whanau, a one hop DHT based routing protocol which exploits social connections between users to construct routing tables which allow for Sybil resilient lookups. The file lookup algorithm suggested offers a significant speed up over the traditional flooding techniques seen in existing peer to peer networks. A major drawback of the approach is that it assumes that honest nodes have more social connections to other honest nodes rather than to Sybil nodes which may not always be the case.

SybilInfer [Danezis & Mittal 2009] offers a decentralized protocol to guard the network against Sybil attacks exploiting the fact that a Sybil attack would interfere with the fast mixing property of social networks. The approach entails a probabilistic model to help tag network nodes as either honest or Sybil wherein each such tag contains an assigned probability, referring to the degree of certainty of the result. The approach suffers from assuming that there is at least one honest node in the network which is known a priori whereas in reality there is always a remote possibility of an attacker mimicking the honest side of the social network as a consequence of which no detector would be able to distinguish the honest region from the corrupt one.

2.4.5 Based on Sybil Behavioral Aspects

[Jyothi & Dharanipragada 2009] have proposed a mechanism that by observing the behaviour of a node by an honest node, the extent of damage Sybils can cause can be limited. The method associates every peer with another non-sybil peer known as SyMon. A given peer's SyMon is chosen dynamically such that the chances of both of them being Sybils are very low. The chosen SyMon is entrusted with the responsibility of moderating the transactions involving the given peer and hence makes it almost impossible for Sybils to compromise the system. The SyMons' feedbacks help new requester peers in verifying the past transaction history of provider peers and hence in identifying honest provider peers that serve good files. Thus, SyMon can help in preventing Sybils from decreasing the content availability of the system. The authors propose secure algorithms to choose the SyMon. Here the method serves its purpose only in the cases where the nodes behaviour can be termed bad by looking at its bad history. But in case of Sybils, the nodes need not necessarily be doing bad activities at the individual identity level.

2.4.6 Incentives

[Margolin & Levine 2007] analyse an economic approach to Sybil attack detection employing a protocol referred to as Informant. Informant uses a Dutch auction technique to determine the minimum possible reward to force the Sybil to reveal itself. The method assumes that there are some nodes in the network called detectives who are not Sybils. These nodes start the protocol offering monetary benefits to nodes if they reveal their Sybil identity. Detectives keep increasing the monetary benefit until it reaches the $\frac{B}{2}$ where B is the detective's monetary benefit for learning about a Sybil relationship. This proposal requires implementation of digital currency. Also the model assumes that Sybil identities are rational.

2.4.7 Sybil Issues Addressed in this Thesis

We have classified and discussed the existing approaches to Sybil attack in the previous sections. Along those lines, the thesis contributes approaches namely in challenge-response, based on behavioural characteristics, and based on psychometric personality tests.

Chapter 3

Algorithms to Improve Search Efficiency

Search is an essential function of peer-to-peer file sharing overlays. The objects are shared by nodes in the network. When a particular node needs a file, first it needs to locate the file and then get the file. Without search functionality, file sharing is not practical.

In this chapter, two methods are proposed to improve the efficiency of search in unstructured peer-to-peer networks.

3.1 Search Efficiency

Search in peer-to-peer networks is different from search in a local repository of documents. Precision and recall are two metrics primarily used to measure the performance of search in a local repository. Suppose a repository having n documents has a set r of relevant documents for a query q. When q is executed, system retrieved a set t of documents. Precision and recall are defined as

$$precision = \frac{|\{r\} \cap \{t\}|}{|\{t\}|} \tag{3.1}$$

$$recall = \frac{|\{r\} \cap \{t\}|}{|\{r\}|} \tag{3.2}$$

Metrics in 3.1 and 3.2 indicate the efficiency of search algorithm. More the precision and recall more efficient is the search algorithm.

In peer-to-peer networks scenario, the search is carried over a network of nodes. Therefore apart from the metrics in 3.1 and 3.2, network performance also needs to be considered. Network resources like bandwidth, storage and computation cycles of nodes are utilised in carrying out the search. The consumption of network resources coupled with characteristics of search results determines the efficiency of a search algorithm. An ideal search algorithm will utilise minimum resources and gives best quality results. The following parameters quantify resource consumption and results quality. Finally we outline a unifying measurement of search efficiency.

- **Coverage:** In an unstructured network, the success of a query depends on its coverage. Larger is the coverage, more is the probability to find the object. Larger coverage also results into large number of results. This is defined as the ratio of number of nodes receiving the query to the total number of nodes in the network. In a network of N nodes, if the query reaches c nodes, then

$$coverage = \frac{c}{N} \tag{3.3}$$

- **Message Count:** Any network protocol requires a set of messages to be transmitted. Here a message is exchanged over a overlay link between two nodes. The generation of such messages has a direct implication on consumption of network resources. The more the messages, the more will be bandwidth consumption and the more will be processing and storage overhead on individual nodes. In a n-node network with average degree d, the average aggregate bandwidth ABW of a search query Q having size $S(Q)$ with TTL k is computed as

$$\begin{aligned} ABW &= [((d-1)+(d-1)^2+\ldots+(d-1)^k)+(1+(d-1) \\ &\quad +(d-1)^2+\ldots+(d-1)^{k-1})]*S(Q) \\ ABW &= \left[(d-1)\tfrac{1-(d-1)^k}{2-d}+\tfrac{1-(d-1)^k}{2-d}\right]*S(Q) \end{aligned}$$

$$ABW = d\left(\frac{1-(d-1)^k}{2-d}\right)*S(Q) \tag{3.4}$$

ABW computed by the eqn. 3.4 gives an approximate because here the duplicate messages, variations in node degree, and responses are not considered.

- **Message Duplication Rate:** As shown in in figure 3.1, duplicate messages get generated during the query propagation. These messages are transmitted over the network but never processed at the receiving ends. They consume bandwidth. Less the number of duplicate messages, more will be the efficiency of the search algorithm.

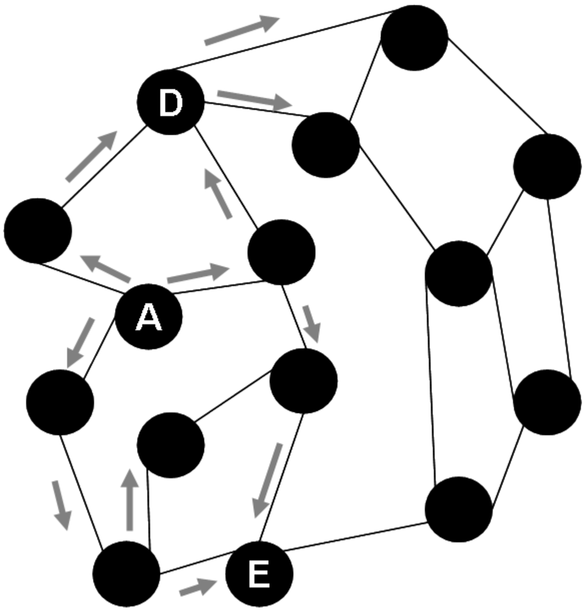


Figure 3.1: Duplicate queries reaching nodes *D* **and** *E* - The query is triggered by *A* reaches *D* and *E* twice although it is not processed

- **Success Rate:** Success rate is how many queries have found the object successfully. In unstructured networks, there is a possibility that the object may not be found although it may be present in the network. This is due to the limited network-coverage of the search. Coverage of flooding is limited by TTL. If there are $|Q|$ queries fired, and if there are only $|SQ|$ successful queries then success rate SR is defined as

$$SR = \frac{|SQ|}{|Q|} \tag{3.5}$$

- **Response Time:** Response time is the time counted from triggering of the query to the time when first response is received. This metric indicates how quickly search could give answers. This metric is related to user satisfaction. Sometimes response time is also measured as the number of hops between requester node and responder node.

- **Query Hits:** Number of query hits is the number of responses received for a single query. The number of responses can at the most be the number of replicas of the object being searched for. This metric gives indication of extent of freedom the user has in choosing where to get the object from.

- **Unifying Metric** All the above metrics give different aspects of search efficiency. [Lin & Wang 2003] defines a single metric for measuring and comparing performance of different search algorithms. For a network of size n, if a query q generates m messages, and returns h number of query hits then query efficiency (QE) of query q is defined as

$$QE_q = \frac{h}{\frac{m}{n}} = \frac{nh}{m} \tag{3.6}$$

Search responsiveness (SP) is defined as ratio of success rate (as in eqn. 3.5) to response time. Response time is proportional to hops between requester and responder. Search responsiveness is defined as

$$SP = \frac{SR}{\sum_{q=1}^{q=|SQ|} \frac{hops_q}{|SQ|}} \tag{3.7}$$

Search Efficiency (SE) is defined as $(QE)_{avg} \times SP$.

$$SE = \sum_{q=1}^{|Q|} \frac{QE_q}{|Q|} \times SP \tag{3.8}$$

This is a relative measurement. It is important to have unifying metric. For example, a search procedure may consume very few messages but the query hits may be very few. This procedure is very efficient but not satisfactory. This unifying metric combines the effect of both, query efficiency and quality.

3.2 Improving Efficiency by Fuzzy Probabilities

In peer-to-peer networks, peers connect to the network through neighbour-peers. Therefore neighbour selection is an important task as a part of joining the network. This is done once at the time of joining the network. There are several criteria by which a peer can select neighbours. But there is another instance where a peer selects a few of its neighbours while searching. This selection of subset of neighbour-peers greatly enhances the efficiency in consuming network resources. In flooding search method, all neighbours are selected to forward the query message. This in-turn is imitated by all the neighbours in the consequent hops. As per bandwidth expression 3.4, query messages increase exponentially with every hop. On other extreme, a random-walk chooses a single neighbour on a random fashion and forwards the query. For a 1000-node network with an average node degree of 8 and a query TTL of 7, flooding generates approximately $8^7 = 2097152$ messages but to reach all 1000 nodes we need only 1000 messages.

Randomwalk generates only a single message in every hop. But Flooding has the advantage of quick coverage of significant network nodes and quick results where as randomwalk is very slow and covers only very small fraction of network. Therefore a balanced approach can blend the advantages of quick coverage and efficient network resource usage. One of the ways of bringing this balance is by being considerate about selecting a subset of neighbours to forward the query. In literature, random selection of neighbours is classified under blind-search schemes and selection based on some specific criteria is classified under intelligent-search schemes.

There are various ways by which a subset of neighbours are selected. Heuristic criteria proposed by [Yang & Garcia-Molina 2002] is shown in table 3.1. Cosine similarity approach is proposed by [Kalogeraki *et al.* 2002]. For a given query q, a peer ranks its neighbours using a cosine similarity function. Peer maintains a profile for each neighbour and records the queries answered by them. When q needs to be forwarded it computes aggregated query similarity between q and the queries answered by its neighbours.

Table 3.1: Criteria for selecting a subset of neighbours - Source: [Yang & Garcia-Molina 2002]

Criteria	Description
$> RES$	Returned the greatest number of results in the past 10 queries
$< TIME$	Had the shortest average time to satisfaction in the past 10 queries
$< HOPS$	Had the smallest average number of hops taken by results in the past 10 queries
$> MSG$	Sent our client the greatest number of messages (all types)
$< QLEN$	Had the shortest message queue
$< LAT$	Had the shortest latency
$> DEG$	Had the highest degree (number of neighbors)

3.2.1 Fuzzy Scheme for Choosing Neighbour-subset

Previous works used criteria based solely on the past performance of the neighbours. Past performance does indicate the node's capability to answer the queries. But recording past performance takes time and it is not immediately applicable to new neighbours. Also for any type of query, using past performance alone may lead to targeting only few nodes making them hotspots and overloaded. In this scheme, we propose a neighbour-subset selection scheme that is based on content classification and a set of heuristics. [Zhao *et al.* 2006] and [Meng *et al.* 2006] analysed the files shared in Gnutella network. One inference from that analysis is that all types of content like audio, video, document, programs are not uniformly shared by nodes. Some nodes share audio more than video, some video more than audio etc. Keeping this in view, a neighbour-subset selection can be based on what type of content the neighbour is sharing. For finer discrimination, each type of file is further classified based on the popularity of the file: popular, normal, rare. The heuristic is that if the query is looking for a rare video file, it is mostly likely that it may be found in a node which shares a large number of video files and out of which large proportion is rare video files. Popularity of a file is decided by the user. The concept of popularity of a file is dependent on a person's perception and it is totally up to the individual to decide whether a file should be regarded as popular or rare or normal. One person may consider a particular file as popular and another may consider it to be normal. This is a subjective choice. Fuzzy logic is useful in representing such ambiguities.

Fuzzy system was first proposed by [Zadeh 1965]. He showed that fuzzy logic can accept values between TRUE and FALSE, unlike classical logic. Elements of a fuzzy set not only represent TRUE or FALSE but also a degree of truth or a degree of falseness. Classical set theory classifies the elements into a crisp set where as fuzzy set theory can classify the elements into a continuous set using the notion of degree of membership. Membership function gives the values in the range $[0, 1]$. We use fuzzy sets to classify content, popularity etc.

3.2.1.1 Neighbour Content Classification

A node in a peer-to-peer network shares variety of content like audio, video, documents etc,. [Meng *et al.* 2006] through their study on Gnutella peer-to-peer network identified the content types the nodes share and the estimated proportions. The table 3.2 shows these content types and the percentage proportion. This information about a node can give indications for what type of content it can be approached. Suppose if a node shares a large number of video files, it is highly probable that the video file another node is searching for may be found here. But this is not guaranteed. It is just an indication. To map this kind of heuristics to peer-to-peer search, we use fuzzy sets to classify the content with some confidence. On the basis of the study of [Meng *et al.* 2006], in every node, the content is classified into Audio, Video, Programs, Archives, Documents, and Pictures. It is upto the individual node to classify the content. Popularity is one of the notions which guide people's decisions and search for an object. For example, generally to look for a rare book, one goes to an established library. To find some book which is very popular, any street shop is fine. These tendencies are also mapped to peer-to-peer search. Each file of a particular content type is classified as popular, normal or rare with some confidence. This way there are totally 18 (6 × 3) categories into which the files shared by a neighbour are classified. A file can be classified into multiple popularity categories as the user thinks it may fit in. For an example, a peer is sharing 110 files. Out of 110 files, audio files are 20, programs are 15, videos are 25, documents are 20, pictures are 25, and archives are 5. These files are further classified as popular, normal and rare files. These statistics are shown in table 3.3.

Table 3.2: Content types and proportion percentage - these statistics are measured over total network

Content Type	Percent of Files
Audio	79%
Program	6%
Video	5%
Document	4%
Picture	4%
Archive	2%

Table 3.3: Example: Files shared by a peer - Popular, normal and rare columns show the number of files under a particular content type

Content Type	Number of Files	Popular	Normal	Rare
Audio	20	10	6	4
Program	15	3	2	10
Video	25	13	5	7
Document	20	20	0	0
Picture	25	20	5	0
Archives	5	3	2	0

3.2.1.2 Fuzzification

Previous section 3.2.1.1 described how a peer classifies its content. The goal of classification is to let the neighbours know the probabilities of finding the contents with it. Each file is classified as Audio-Rare, Audio-Normal, Audio-Popular, Video-Rare, Video-Normal, Video-Popular etc. Given that a neighbour is searching for Video-Rare file, what is the probability that such a file can be found in this peer. To answer such questions, a peer computes the probabilities for all 18 categories. There are simple ways of computing the probabilities but here the human search tendencies needs to be properly represented in the probability value. Fuzzy sets help in capturing the human judgements in quantifying values. This section describes the fuzzy variables, fuzzy sets and their membership functions used in the proposed search scheme.

There are three fuzzy input variables, namely the number of files in a content category, the percentage of files in a popularity subcategory and the probability of finding the content in neighbours.

Number of files in a content category (nof): The input variable nof consists of three fuzzy sets, i.e., *large*, *medium*, and *small*. The fuzzifiers used are triangular and trapezoidal membership functions. The membership function associated with each fuzzy set is defined as

$$\mu_{small}^{nof} = \begin{cases} 1 & \text{if } nof < 20 \\ \frac{80-nof}{60} & \text{if } nof \geq 20 \text{ and } nof \leq 80 \\ 0 & \text{if } nof > 80 \end{cases} \tag{3.9}$$

$$\mu_{medium}^{nof} = \begin{cases} 0 & \text{if } nof < 20 \\ \frac{nof-20}{30} & \text{if } nof \geq 20 \text{ and } nof \leq 50 \\ \frac{80-nof}{30} & \text{if } nof \geq 50 \text{ and } nof \leq 80 \\ 0 & \text{if } nof > 80 \end{cases} \tag{3.10}$$

$$\mu_{large}^{nof} = \begin{cases} 0 & \text{if } nof < 20 \\ \frac{nof-20}{60} & \text{if } nof \geq 20 \text{ and } nof \leq 80 \\ 1 & \text{if } nof \geq 80 \text{ and } nof \leq 100 \\ 0 & \text{if } nof > 100 \end{cases} \tag{3.11}$$

These membership functions are determined by intuition and experience. A graphical presentation of membership function of nof is shown in figure 3.2. For example, to calculate the probability of Video-Rare, nof (as listed in table 3.3) is 25. From equations 3.11, 3.10, and 3.9, the values for each fuzzy set are determined as

$\mu_{small}^{nof}(25) = 0.92$ $\mu_{medium}^{nof}(25) = 0.17$ $\mu_{large}^{nof}(25) = 0.084$

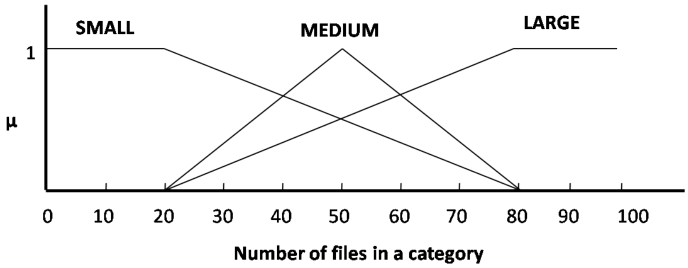


Figure 3.2: Membership function for number of files (*nof*) - fuzzy sets *small* and *large* have trapezoidal fuzzifiers and *medium* has triangular membership function

Percentage of files in a popularity subcategory(*pof*): The input variable pof is the percentage proportion of number of files of a popularity sub-category in a category. Suppose we want a very old book whose publication had stopped. Its more probable that we will find it in a library which keeps old books rather than a book shop though both of them contains almost the same number of books. The probability of finding a rare book is more in a library since the library has more number of such rare books. The tendency to share or liking for certain type of files is represented by percentage proportion. The input variable pof consists of three fuzzy sets, i.e., *high*, *medium*, and *low*. The membership function associated with each fuzzy set is defined as

$$\mu^{pof}_{low} = \begin{cases} 1 & \text{if } pof < 20 \\ \frac{80-pof}{60} & \text{if } pof \geq 20 \text{ and } pof \leq 80 \\ 0 & \text{if } pof > 80 \end{cases} \tag{3.12}$$

$$\mu^{pof}_{medium} = \begin{cases} 0 & \text{if } pof < 20 \\ \frac{pof-20}{30} & \text{if } pof \geq 20 \text{ and } pof \leq 50 \\ \frac{80-pof}{30} & \text{if } pof \geq 50 \text{ and } pof \leq 80 \\ 0 & \text{if } pof > 80 \end{cases} \tag{3.13}$$

$$\mu^{pof}_{large} = \begin{cases} 0 & \text{if } pof < 20 \\ \frac{pof-20}{60} & \text{if } pof \geq 20 \text{ and } pof \leq 80 \\ 1 & \text{if } pof \geq 80 \text{ and } pof \leq 100 \\ 0 & \text{if } pof > 100 \end{cases} \tag{3.14}$$

A graphical presentation of membership function of pof is shown in figure 3.3. For example, to calculate the probability of Video-Rare, pof (as listed in table 3.3) for rare files is $\frac{7}{25} \times 100 = 28\%$. From equations 3.14, 3.13, and 3.12, the values for each fuzzy set are determined as

$\mu^{pof}_{low}(28) = 0.87$ $\mu^{pof}_{medium}(28) = 0.27$ $\mu^{nof}_{high}(28) = 0.14$

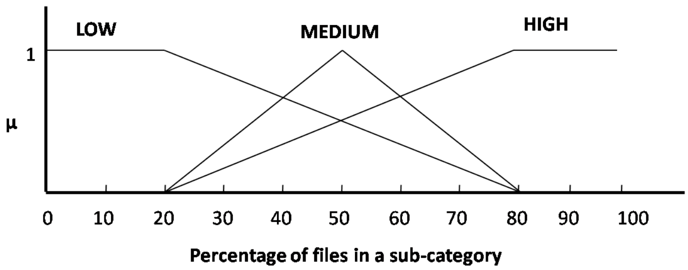


Figure 3.3: Membership function for percentage of popularity sub-category files (*pof*) - fuzzy sets *low* and *high* have trapezoidal fuzzifiers and *medium* has triangular membership function

Probability ($\leq$ 1) of finding the same content in neighbours(*ngprob*): The input variable *ngprob* is the probability of finding the same category-subcategory content in the neighbours. If the neighbours have different probabilities, the maximum is assigned to this variable. Purpose of this variable is to embed the next-hop feasibility in the current hop's probability. The input variable *ngprob* consists of three fuzzy sets, i.e., *high*, *medium*, and *low*. The membership function associated with each fuzzy set is defined as

$$\mu_{low}^{ngprob} = \begin{cases} 1 & \text{if } ngprob < 0.2 \\ \frac{0.8-ngprob}{0.6} & \text{if } ngprob \geq 0.2 \text{ and } ngprob \leq 0.8 \\ 0 & \text{if } ngprob > 0.8 \end{cases} \tag{3.15}$$

$$\mu_{medium}^{ngprob} = \begin{cases} 0 & \text{if } pof < 0.2 \\ \frac{ngprob-0.2}{0.3} & \text{if } ngprob \geq 0.2 \text{ and } ngprob \leq 0.5 \\ \frac{0.8-ngprob}{0.3} & \text{if } ngprob \geq 0.5 \text{ and } ngprob \leq 0.8 \\ 0 & \text{if } ngprob > 0.8 \end{cases} \tag{3.16}$$

$$\mu_{large}^{ngprob} = \begin{cases} 0 & \text{if } pof < 0.2 \\ \frac{ngprob-0.2}{0.6} & \text{if } ngprob \geq 0.2 \text{ and } ngprob \leq 0.8 \\ 1 & \text{if } ngprob \geq 0.8 \text{ and } ngprob \leq 1 \\ 0 & \text{if } ngprob > 1 \end{cases} \tag{3.17}$$

A pictorial presentation of membership function of *ngprob* is shown in figure 3.4. For example, to calculate the probability of Video-Rare, *ngprob* of five neighbours is taken as $max\{0.2, 0.34, 0.13, 0.67, 0\} = 0.67$. From equations 3.17, 3.16, and 3.15, the values for each fuzzy set are determined as

$\mu_{low}^{ngprob}(0.67) = 0.23$ $\quad\mu_{medium}^{ngprob}(0.67) = 0.44$ $\quad\mu_{high}^{ngprob}(0.67) = 0.78$

3.2.1.3 Output Fuzzy Set

The probability of finding category-subcategory content in a node is considered to be the output fuzzy variable. The output fuzzy variable *probability* has five fuzzy sets, i.e., *verylow*, *low*, *medium*, *high*, and *veryhigh*. The universe of discourse for each member of the fuzzy set is

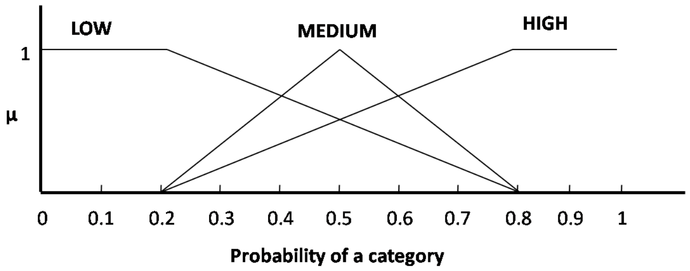


Figure 3.4: Membership function for of finding the probability of the same content in neighbours (*ngprob*) - fuzzy sets *low* and *high* have trapezoidal fuzzifiers and *medium* has triangular membership function

- Very Low $[0, p]$
- Low $[q, r]$
- Medium $[s, t]$
- High $[u, v]$
- Very High $[w, 1]$

where p, q, r, s, t, u, v, w are real numbers between 0 and 1. The membership function for *probability* is pictorially shown in 3.5.

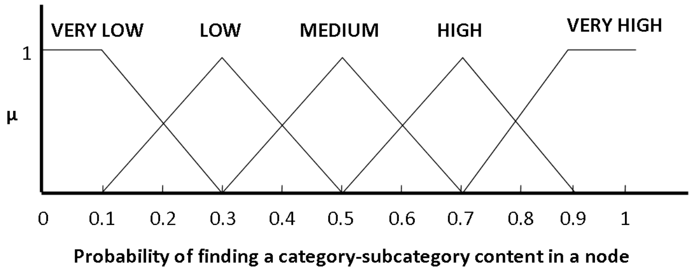


Figure 3.5: Membership function for fuzzy output variable (*probability*) - fuzzy sets *verylow* and *veryhigh* have trapezoidal fuzzifiers and others have triangular membership function

3.2.1.4 Knowledge Base

Rules are designed based on human logic to find the output from the conditions of input variables. They are arrived at by experience. For example, if the number of video files is large and the percentage of the files with 'rare' popularity rating is medium and the probability of finding the 'video-rare' content in neighbours is low then the probability of finding a 'rare video' file in that peer is high. There are several rules like this. They are all put into tabular format as shown in table 3.4. These 27 rules are used to determine the value of output fuzzy variable. The fuzzy output is defuzzified to get the actual probability value.

Table 3.4: Fuzzy rules - nof, pof, and $ngprob$ are input fuzzy variables and $probability$ is the output fuzzy variable

Rule	nof	pof	$ngprob$	$probability$
1	small	low	low	very low
2	small	medium	low	very low
3	small	high	low	low
4	medium	low	low	low
5	medium	medium	low	medium
6	medium	high	low	medium
7	large	low	low	high
8	large	medium	low	high
9	large	high	low	very high
10	small	low	medium	very low
11	small	medium	medium	very low
12	small	high	medium	low
13	medium	low	medium	medium
14	medium	medium	medium	medium
15	medium	high	medium	medium
16	large	low	medium	high
17	large	medium	medium	very high
18	large	high	medium	very high
19	small	low	high	very low
20	small	medium	high	low
21	small	high	high	low
22	medium	low	high	medium
23	medium	medium	high	medium
24	medium	high	high	high
25	large	low	high	high
26	large	medium	high	very high
27	large	high	high	very high

3.2.1.5 Rule Implication

A rule contains antecedents (inputs) and consequent (output) actions. Rule gives the linguistic mapping between these two sets. To map on degree of membership between input and output fuzzy sets, Mamdani implication rule [Mamdani 1977] is used. For the input fuzzy set A and output fuzzy set B, Mamdani implication rule is stated in eqn. 3.18.

The implication result is a fuzzy set which is a minimum of μ_A and μ_B.

$$\phi[\mu_A(x), \mu_B(y)] \equiv \mu_A(x) \wedge \mu_B(y) \tag{3.18}$$

Example for calculating Video-Rare probability: Consider Rule 1 in table 3.4. It states that if the number of video files are small, and the percentage of rare video files is low and the probability of finding it in neighbour is also low then the probability of finding a rare-video in this node is very low. In this rule there are three input fuzzy sets and one output fuzzy set. First, all three input fuzzy sets need to be combined into one. They are connected by 'and' connective. The 'and' connective is replaced by intersection operator. Intersection of fuzzy sets A and B is the minimum degree of memberships of sets A and B. It is stated in eqn. 3.19.

$$\begin{aligned} \mu_{A\cap B}(x) &= \mu_A(x) \wedge \mu_B(x) \\ &= \min(\mu_A(x), \mu_B(x)) \end{aligned} \tag{3.19}$$

The resultant antecedent value for Rule 1 is calculated as

$$\begin{aligned} &= \mu^{nof}_{small}(25) \wedge \mu^{pof}_{low}(28) \wedge \mu^{ngprob}_{low}(0.67) \\ &= min\left\{0.92, 0.87, 0.23\right\} \\ &= 0.23 \end{aligned}$$

Now Mamdani implication rule (eqn. 3.18) is applied to antecedent and consequent fuzzy sets of the rule.

$$\begin{aligned} &\equiv \mu_A(x) \wedge \mu_B(y) \\ &\equiv \min\left\{0.23, \mu^{probability}_{verylow}\right\} \end{aligned}$$

The resultant antecedent membership value 0.23 truncates the membership of *verylow* output fuzzy set along the ordinate axis. This is graphically presented in figure 3.6. The same way, for the 'video-rare' example, other 26 rules need to be applied. While applying rules, the same fuzzy set may have two different Mamdani rule implications. For example, for rule 1 and rule 19, the output fuzzy set *verylow* receives two implications. This is graphically presented in 3.7.In such cases, the output is aggregated using union operator.

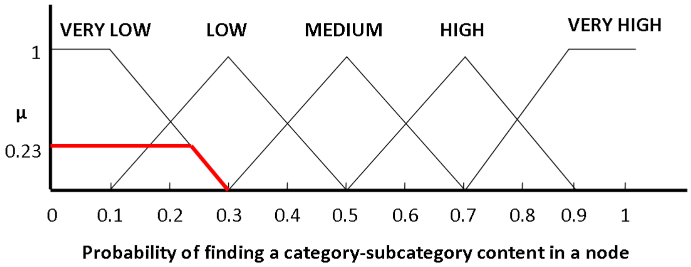


Figure 3.6: Mamdani implication rule applied to Rule 1 - Red line is the minimum value along the ordinate axis

Table 3.5 shows the application of Mamdani implication rule to all rules for the 'rare-video' example. The final aggregated values for each fuzzy set is listed in table 3.6. The implications are graphically presented in figure 3.8.

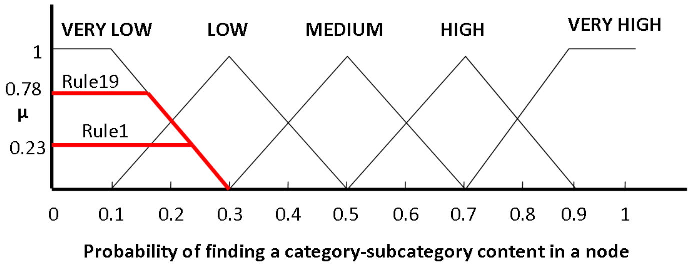


Figure 3.7: Mamdani implication rule applied to Rule 1 and Rule 19 - final output is the union of the output by Rule 1 and Rule 19

3.2.1.6 Defuzzification

Fuzzified inputs are transformed into fuzzy outputs. The process of deriving a scalar or crisp value from the fuzzified outputs is called defuzzification. The final fuzzy output aggregate is shown in figure 3.8. The centre of gravity method is the most commonly used method to derive crisp value. Centre of gravity method was proposed by [Sugeno 1985]. The defuzzified value is obtained by

$$x^* = \frac{\int \mu_i(x)xdx}{\int \mu_i(x)dx} \tag{3.20}$$

where x^* is the defuzzified output, $\mu_i(x)$ is the aggregated membership function and x is the output variable. For the aggregated output shown in figure 3.8, the defuzzified output

Table 3.5: Mamdani implication rule applied to all rules - actual implication is on the fuzzy set membership. The last column shows the truncating value along the ordinate axis for fuzzy output set *probability*

Rule	*nof*	*pof*	*ngprob*	upper bound
1	0.92	0.87	0.23	0.23
2	0.92	0.27	0.23	0.23
3	0.92	0.14	0.23	0.14
4	0.17	0.87	0.23	0.17
5	0.17	0.27	0.23	0.17
6	0.17	0.14	0.23	0.14
7	0.084	0.87	0.23	0.084
8	0.084	0.27	0.23	0.084
9	0.084	0.14	0.23	0.084
10	0.92	0.87	0.44	0.44
11	0.92	0.27	0.44	0.27
12	0.92	0.14	0.44	0.14
13	0.17	0.87	0.44	0.17
14	0.17	0.27	0.44	0.17
15	0.17	0.14	0.44	0.14
16	0.084	0.87	0.44	0.084
17	0.084	0.27	0.44	0.084
18	0.084	0.14	0.44	0.084
19	0.92	0.87	0.78	0.78
20	0.92	0.27	0.78	0.27
21	0.92	0.14	0.78	0.14
22	0.17	0.87	0.78	0.17
23	0.17	0.27	0.78	0.17
24	0.17	0.14	0.78	0.14
25	0.084	0.87	0.78	0.084
26	0.084	0.27	0.78	0.084
27	0.084	0.14	0.78	0.084

Table 3.6: Aggregated fuzzy outputs - the outputs for different rules are aggregated using union operator

Output Fuzzy Set	**Upper Bound**
Very Low	0.78
Low	0.27
Medium	0.17
High	0.14
Very High	0.084

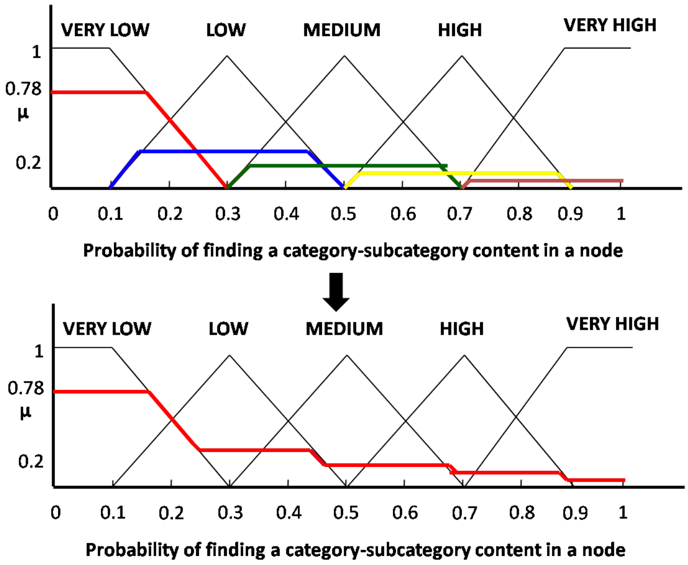


Figure 3.8: Aggregated fuzzy outputs of all rules - final aggregation is also shown in the bottom portion

can be computed as

$$
\begin{aligned}
x^* &= \frac{\int_0^{0.144} 0.78xdx + \int_{0.144}^{0.246}(-5x+1.5)xdx + \int_{0.246}^{0.446} 0.27xdx + \int_{0.446}^{0.466}(-5x+2.5)xdx + \int_{0.466}^{0.666} 0.17xdx +}{\int_0^{0.144} 0.78dx + \int_{0.144}^{0.246}(-5x+1.5)dx + \int_{0.246}^{0.446} 0.27dx + \int_{0.446}^{0.466}(-5x+2.5)dx + \int_{0.466}^{0.666} 0.17dx +} \\
&\quad \frac{+ \int_{0.666}^{0.672}(-5x+3.5)xdx + \int_{0.672}^{0.872} 0.14xdx + \int_{0.872}^{0.8832}(-5x+4.5)xdx + \int_{0.8832}^{1} 0.084xdx}{+ \int_{0.666}^{0.672}(-5x+3.5)dx + \int_{0.672}^{0.872} 0.14dx + \int_{0.872}^{0.8832}(-5x+4.5)dx + \int_{0.8832}^{1} 0.084dx} \\
x^* &= \frac{0.008+0.01+0.0186+0.002+0.019+0.0006+0.02+0.001+0.009}{0.11+0.053+0.054+0.0044+0.034+0.00093+0.028+0.001+0.0098} \\
x^* &= \frac{0.09}{0.298} \\
x^* &= 0.3
\end{aligned}
$$

This calculation is for the category 'Video-Rare'. It says that a rare video file can be found in this node with probability of 0.3. There are 17 more categories for which the same procedure is applied to obtain the probabilities. Every node maintains 6 × 3 matrix containing probability values for 18 categories. This matrix is broadcast to neighbours upon any changes.

3.2.2 Search Algorithm

Previous section described how a node classifies its content and assigns probabilities to content classes. In search process, these probabilities are used to direct the queries. There are two actors in this process, i.e.,

- Query Processing Node

- Neighbour Node

Query Processing Node: This node is any node which processes the query that is on its way. If a node *n* has *ng* neighbours, then it would maintain *ng* 6 × 3 probability matrices in addition to its own. When a search query arrives at its door step, it will execute steps as shown in the algorithm given in figure 3.9. When the node joins the network, it gets the probability tables from its neighbours.

```
Input: Query q
if TTL(q)=0 then
    discard query;
    return;
end
if q is already processed then
    discard query;
    return;
end
foreach file in FileSet do
    /* check whether the file is found locally                */
    $hits \leftarrow \phi$;
    if keywords(q) match keywords(file) then
        $hits \leftarrow hits \cup file$;
    end
end
if $hits \neq \phi$ then
    send hits back to requester;
else
    continue;
end
foreach neighbour ng in neighbourSet do
    /* probability tables are stored in local cache           */
    $probTable \leftarrow getProbabilityTable(ng)$;
    $prob[ng] \leftarrow findProbability(probTable, fileType(q), popularity(q))$;
end
$sortDescendingOrder(prob)$;
$ng \leftarrow getTopEntry(prob)$;
$TTL(q) \leftarrow TTL(q) - 1$;
Forward Query q to ng ;
return;
```

Figure 3.9: Procedure: *ProcessSearchQuery*()

Neighbour Node: Every neighbour node maintains its own probability tables. Whenever there is a content change in the neighbour node it updates the probability values and broadcasts probability table to all its neighbours. The steps in this process is shown in the algorithm given in figure 3.10.

```
foreach category c in 6 ContentTypes do
    foreach subcategory s in 3 Popularities do
        Compute $nof$, $pof$, $ngProb$;
        Fuzzify input variables $nof$, $pof$, $ngProb$;
        foreach rule r in FuzzyRuleSet do
            Compute resultant antecedent membership function for $r$;
            Apply Mamdani Implication Rule on $r$;
        end
        Aggregate the fuzzy outputs for all rules;
        $prob[c, s] \leftarrow$ Defuzzified crisp value using centre of gravity method;
    end
end
foreach neighbour ng in NeighbourSet do
    send $prob$ to $ng$;
end
return;
```

Figure 3.10: Procedure: $UpdateProbabilityTable()$

3.2.3 Experiment Setup

A discrete-event simulator is developed to perform simulations.

3.2.3.1 Simulator Model

The simulator model is shown in figure 3.11. Nodes are modelled as objects who have references for the neighbour nodes. Every node has an incoming queue that accumulates the messages added by neighbour nodes. The processing of these messages is handled by an individual thread. Each node object is handled by a separate thread. The message passing is done by directly adding the message to the incoming queue of the destination node. Each node runs in an infinite loop processing the messages in the queue. The end of simulation is indicated by the Simulator thread by setting a loop-guard parameter to true. The messages are categorized into query, and query hit. The messages are treated differently according to their type. If the message is a query it is cloned and forwarded to neighbours.

Simulator thread reads the topology and accordingly builds the whole network. The bandwidth and latency values are assigned to the links. It also distributes the objects to nodes according to the plan given. Simulator thread fires the queries in sequential way on behalf of the nodes. It puts the query messages into the queue of the corresponding node. This model resembles the real world network very closely. The advantage of this model is that one can record time based observations like traffic growth w.r.t time. It also gives fine control on logging, and giving statistics summaries.

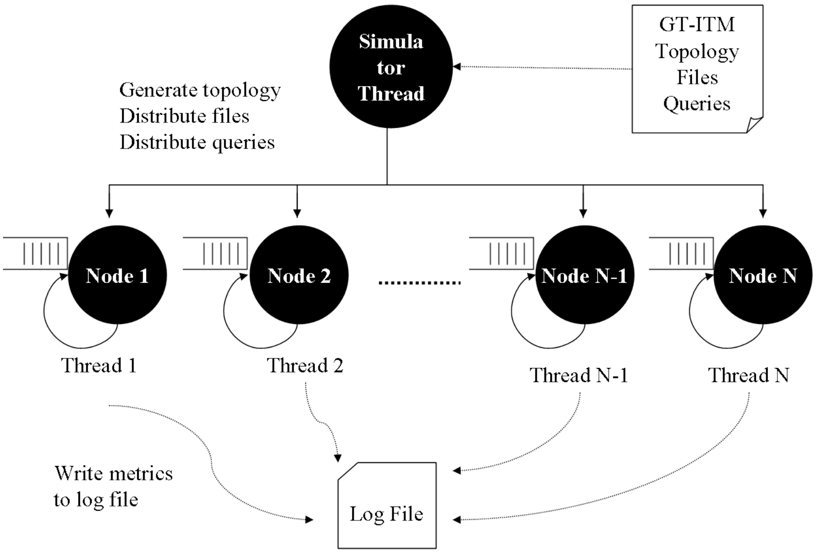


Figure 3.11: Simulator Model - each node is an object and a queue is attached to every node

3.2.3.2 Simulation Setup

A 1000-node topology, generated by GT-ITM[Zegura *et al.* 1996] is used. Simulator builds a network of 1000 nodes with average degree of 9.34. There are 150 distinct objects in the network. Each object is replicated in proportion to its popularity following zipf distribution [Chu *et al.* 2002]. If n objects are ranked from 1 to n by their popularity, then the number of replicas r of object i with rank k are calculated as

$$r_i \propto \frac{1}{k^{\alpha}} \tag{3.21}$$

α is taken as 0.83. There is a total of 4756 objects spread across the network. 3008 queries are fired for objects in the network. The proportion of queries for an object is proportional to the popularity of the object, reflecting the real situation. Simulation is done for different neighbour-subset selection strategies proposed in the literature. In all cases, query propagation is done through 20 walkers, each with TTL of 1024.

3.2.4 Result Analysis

The purpose of carrying out this simulation was to compare our approach with other neighbour-subset selection approaches proposed in the literature. Apart from the fuzzy based approach developed herein, random selection, cosine similarity, maximum node degree, maximum messages, maximum results, and minimum hops were considered for simulation and cross comparison. All approaches were simulated with a common simulation environment. Search performance of different approaches was compared on the basis of the metrics mentioned in the section 3.1. The search statistics were collected from log files. They are displayed in table 3.7.

Table 3.7: Search performance metrics computed for various approaches - these statics are obtained from simulations done in the same environment

	Random Selection	Cosine Similarity	Maximum Results	Maximum Node degree	Minimum Hops	Maximum Messages	Fuzzy Based Probability
Number of nodes	1000	1000	1000	1000	1000	1000	1000
Total queries fired	3008	3008	3008	3008	3008	3008	3008
Total query hits	17195	19508	15771	11853	15612	17885	18438
Successful queries	2941	2905	2851	2577	2856	2954	2929
Total messages	354750	403424	302030	179165	300156	381451	224149
Average Hops from HitNode	5.51	3.296	4.734	2.93	4.62	6.07	3.488
Duplicate Messages	11051	8744	12494	16392	12648	10520	9822
Avg Query efficiency	1.61	1.61	1.74	2.2	1.73	1.56	2.73
Search Responsiveness	17.74	29.3	20.02	29.24	20.55	16.18	27.92
Search efficiency	28.56	47.17	34.83	64.33	35.55	25.24	76.22

3.2.4.1 Query Efficiency

Query efficiency is measured by the following parameters. They indicate consumption of network resources.

- **Message Generation:** From the table 3.7, number of messages generated are largest in cosine similarity approach and smallest in maximum-node-degree approach. Larger are the number of messages, greater is the network bandwidth consumed. The number of messages generated by Fuzzy-probabilities approach is nearly half of that of cosine similarity approach. Message generation is proportional to number of hops taken by the query. In cosine similarity approach, the object is found far away from the requester node and very near by in maximum-node-degree approach.

- **Duplicate Messages:** One of the factors that effect performance of the search algorithm is duplicate messages. A duplicate message is a message that is forwarded to a neighbour which processed the same message before. A large number of duplicate messages indicate that there is some problem with the query propagation. The messages would have been directed in a better way. Duplicate message percentages for these approaches are presented in figure 3.12.

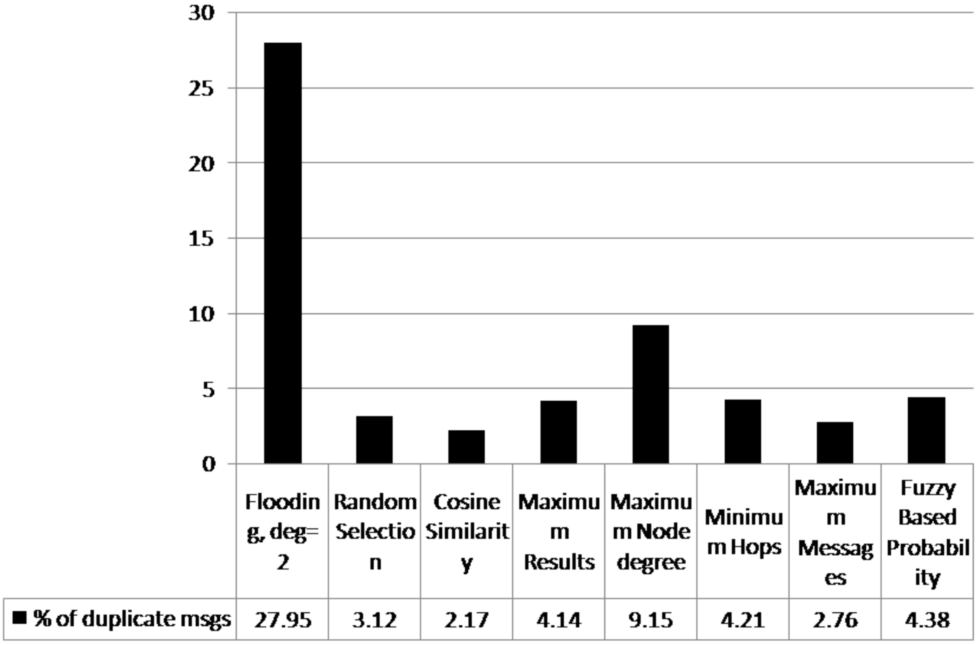


Figure 3.12: Percentages of duplicate messages of neighbour subset selection approaches - Flooding is not a neighbour subset selection approach. It is given here as a contrast.

From the figure it can be observed that the highest duplication percentage is of flooding. Among the neighbour subset selection approaches, highest percentage is of maximum node degree 9.15% and the least is of cosine similarity approach 2.17%. In cosine similarity approach, the forwarding of the query is based on how closely the keywords match with the profile of the neighbour. Therefore every query is forwarded according to the characteristics of the query. But in maximum-node-degree approach, every query is forwarded to the neighbour with maximum degree. Therefore there is high probability that queries will reach the same node again. In fuzzy-based-probability approach, 4.38% of messages are getting wasted. Although the query propagation is based on content type, but not by the actual keyword match as in the case of cosine-similarity approach.

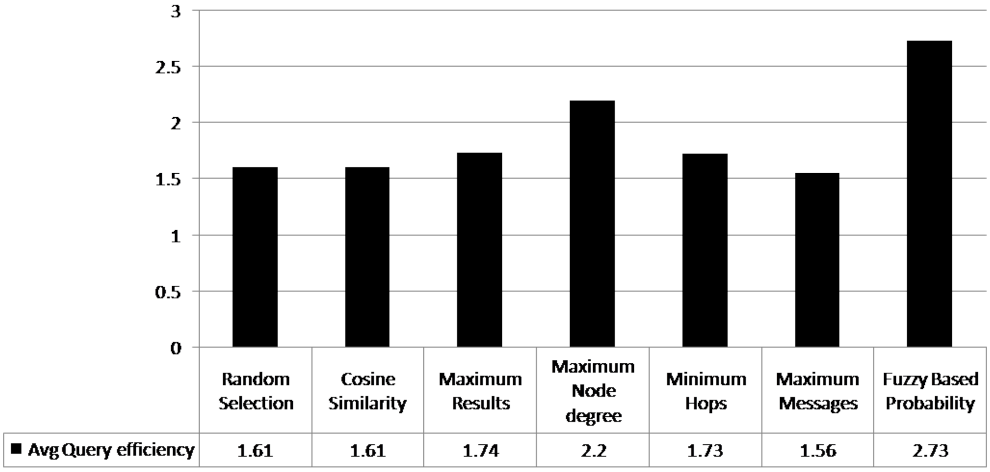


Figure 3.13: Average query efficiencies of neighbour subset selection approaches - efficiency value doesn't correspond to percent.

Different approaches have advantages in different metrics. Query efficiency is measured by eq. 3.6. Query efficiencies for all approaches are depicted in figure 3.13. Query efficiency is maximum in fuzzy-probability approach.

3.2.4.2 Search Responsiveness

Search responsiveness is measured by the following parameters. They measure user satisfaction.

- **Success Rate:** Success rate is defined by eq. 3.5. It is maximum in maximum-messages approach and minimum in maximum-node-degree approach. Fuzzy-probabilities approach has success rate near to best case. Success rate indicates ability to find the object even if it is in far away place.

- **Hop Count:** Hop count is measured as the number of hops between the requester node and the hit node. Hop count is small, it means the response is very quick. Smallest hop count is found in maximum-node-degree approach and largest is found in maximum-messages approach. Fuzzy approach is near the best case. This is graphically presented in figure 3.14. It means that maximum-node-degree approach is finding objects in the shortest path. This also means that minimum number of messages are required to reach the hitnode.

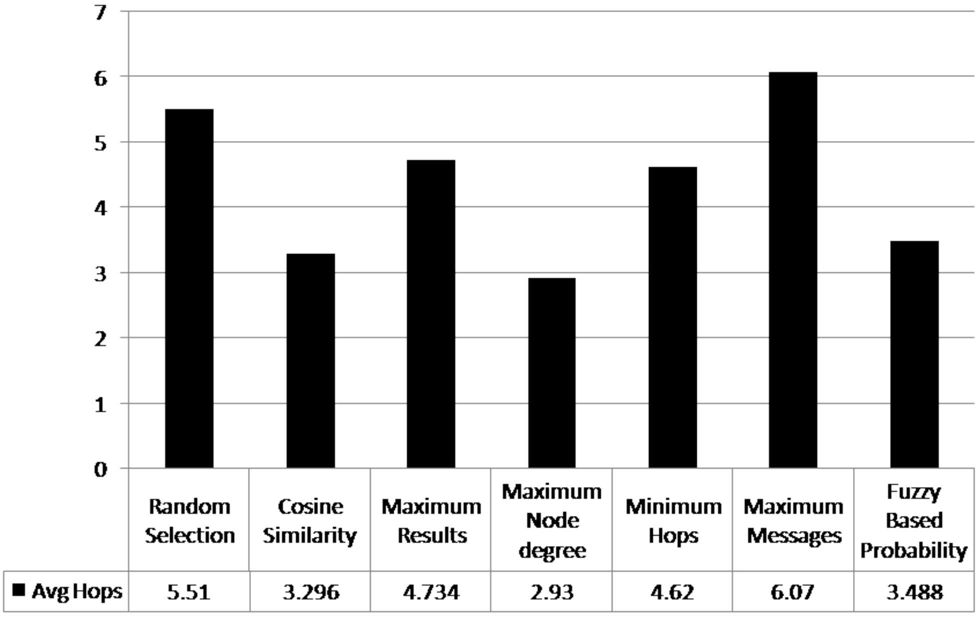


Figure 3.14: Average hops between the requesting node and the hit node in neighbour subset selection approaches - hops indicate the response time

- **Query Hits:** Query hits are the number of responses received from network for a query. User can select a suitable response and initiate the transfer of data. Cosine-similarity model has maximum number of responses returned.

  Search responsiveness is given by eq. 3.7. These values are depicted in figure 3.15. Best response is offered by cosine-similarity, maximum-node-degree approaches and fuzzy-probability approach.

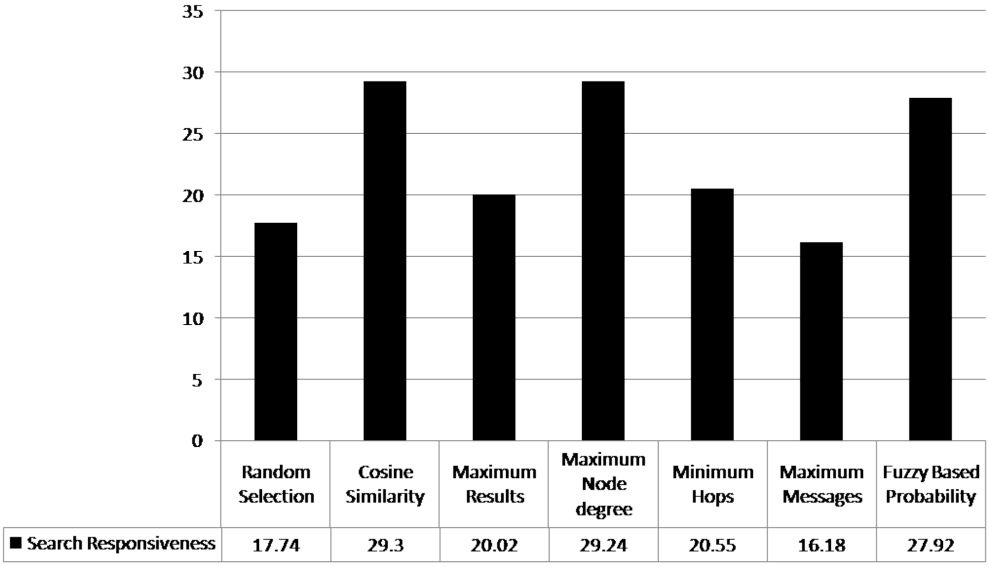


Figure 3.15: Search responsiveness of neighbour subset selection approaches - responsiveness value doesn't correspond to percent. It is merely a metric for comparison and doesn't stand on its own.

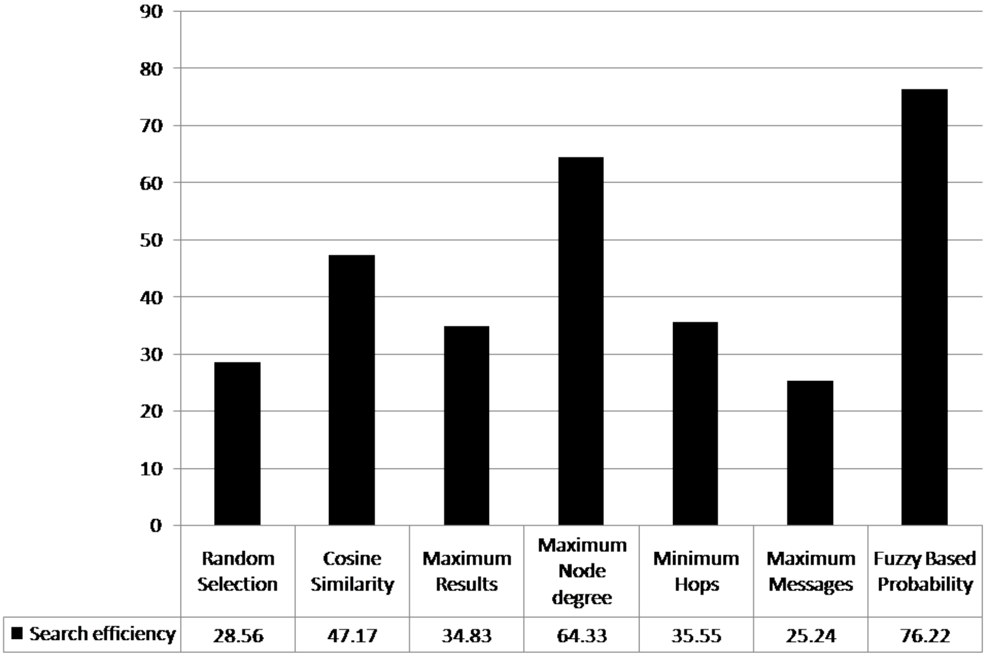
3.2.4.3 Search Efficiency

Each approach is performing good in some parameter and performing bad in some parameter. Search efficiency is measured by the eq. 3.8. These values are computed from table 3.7 and plotted in figure 3.16. The maximum search efficiency is found in fuzzy-probability approach. It has 18.5% increase over maximum-node-degree approach.

3.2.4.4 Load Distribution on Neighbours

One of the implications of neighbour selection approaches is the load they incur on the neighbours. If the same set of neighbours are chosen frequently to forward the queries, it will overburden them. Comparison of the load per node in three approaches is presented in figure 3.17.

The graph presents message generation CDF curves of maximum-node-degree, cosine-similarity and fuzzy-probability approach. It can be noted that cosine-similarity has fastest growing CDF and maximum-node-degree approach has the slowest growing curve upto some node. Cosine-similarity approach generated largest number of messages. Maximum-node-degree approach generated smallest number of messages. Another important thing to look at is the pattern of load per node. In maximum-node-degree ap-



	Random Selection	Cosine Similarity	Maximum Results	Maximum Node degree	Minimum Hops	Maximum Messages	Fuzzy Based Probability
■ Search efficiency	28.56	47.17	34.83	64.33	35.55	25.24	76.22

Figure 3.16: Search efficiencies of neighbour subset selection approaches - efficiency value doesn't correspond to percent. It is merely a metric for comparison and doesn't stand on its own.

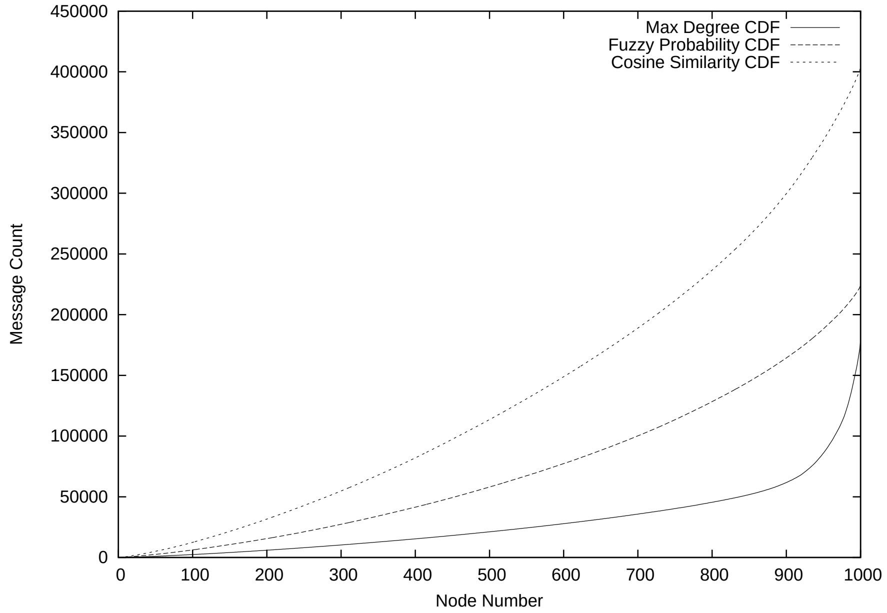


Figure 3.17: CDF of messages per node for top three neighbour subset selection approaches - CDF means Cumulative Distribution Function

proach there is sudden increase in the slope after node number 900. A closer look at this reveals that 92% of the nodes processed only 38% of the traffic and 8% of the nodes process 62% of the traffic. This means that some of the neighbours are overloaded than others. There will some load imbalance because nodes are heterogeneous in their capacities. Some have better capabilities than others. But here the load distribution is greatly skewed. In cosine similarity approach, 8% of nodes processed 21.6% of the traffic whereas in fuzzy-probability approach 8% of nodes processed 22.8% of the query traffic.

3.2.4.5 Hybrid Approach

The disadvantage of maximum-node-approach is that it creates a load-imbalance on neighbours, high percentage of duplicate messages, low success rate and it has low query efficiency. Advantages of this approach are targets are found in shortest path, and has got high search responsiveness. The disadvantage of fuzzy-based approach is that it is not able to find the objects in short paths and its responsiveness is also low. The advantage of this approach is that it has got high success rate and the highest query efficiency. Combining the two approaches can result in an hybrid approach to neighbour selection. The hybrid criteria for choosing the neighbour j is computed as

$$criteria_j = w1 \times \frac{nodedegree_j}{d_{max}} + w2 \times fp \tag{3.22}$$

where $w1$ and $w2$ are weights for degree of neighbour and fuzzy probability of neighbour respectively. d_{max} is the highest node degree in the network and fp is the fuzzy probability of the neighbour.

As it is expected, this approach has got higher returns. The results are summarised in table 3.8. Query efficiency, search responsiveness and search efficiency are shown in figures 3.18, 3.19 and 3.20. The two approaches are combined at various proportions of $w1$ and $w2$ to get a hybrid metric. There are improvements in both ways, i.e., query efficiency and search responsiveness. The maximum search efficiency is found when they are combined at 36-64 proportion, i.e., 36% weight given to maximum-node-degree and 64% given to fuzzy-probability approach. The load imbalance also is reduced as depicted in figure 3.21. In Hybrid approach, 8% of the nodes processed 38% of the query traffic in the network as compared to 62% in the case of maximum-node-degree approach..

Table 3.8: Search performance metrics computed for hybrid approach - these statics are obtained from simulations done in the same environment

	Only Node Degree (100-0)	60-40	50-50	40-60	37-63	36-64	35-65	33-67	30-70	Only Fuzzy Prob (0-100)
No of nodes	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
Total query hits	11853	14050	14987	16118	16398	16451	16534	16720	17115	18438
Successful queries	2577	2731	2776	2833	2836	2853	2856	2863	2856	2929
Total msgs	179165	182179	184681	190604	192137	192074	193937	196142	202150	224149
Total queries	3008	3008	3008	3008	3008	3008	3008	3008	3008	3008
Avg Hops	2.93	2.98	3	3.043	3.058	3.0356	3.071	3.133	3.19	3.488
Avg Query efficiency	2.2	2.56	2.7	2.81	2.84	2.85	2.83	2.83	2.81	2.73
Search Responsiveness	29.24	30.47	30.76	30.95	30.83	31.24	30.92	30.38	29.76	27.92
Search efficiency	64.33	78	83.05	86.97	87.56	89.03	87.5	85.98	83.63	76.22

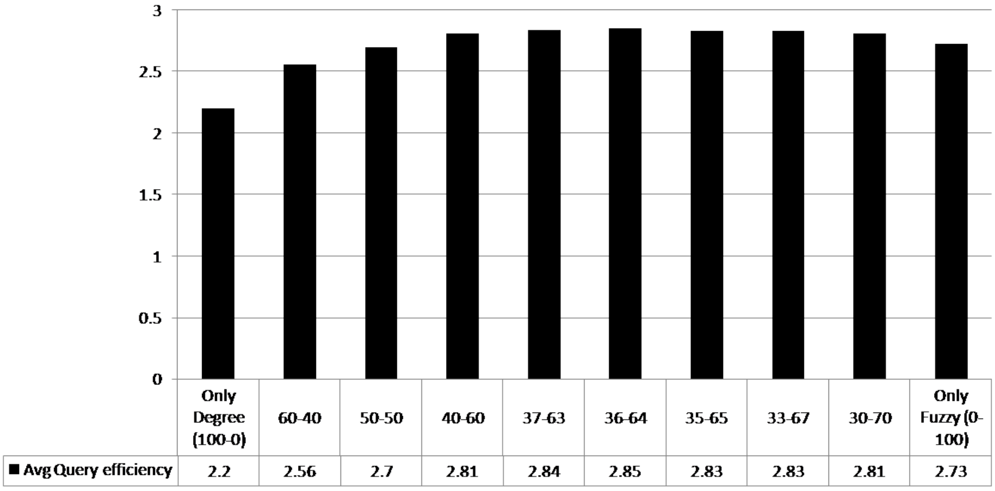


Figure 3.18: Average query efficiencies of the hybrid approach - 60-40 indicates 60% weight given to maximum-node-degree and 40% weight given to fuzzy-probability

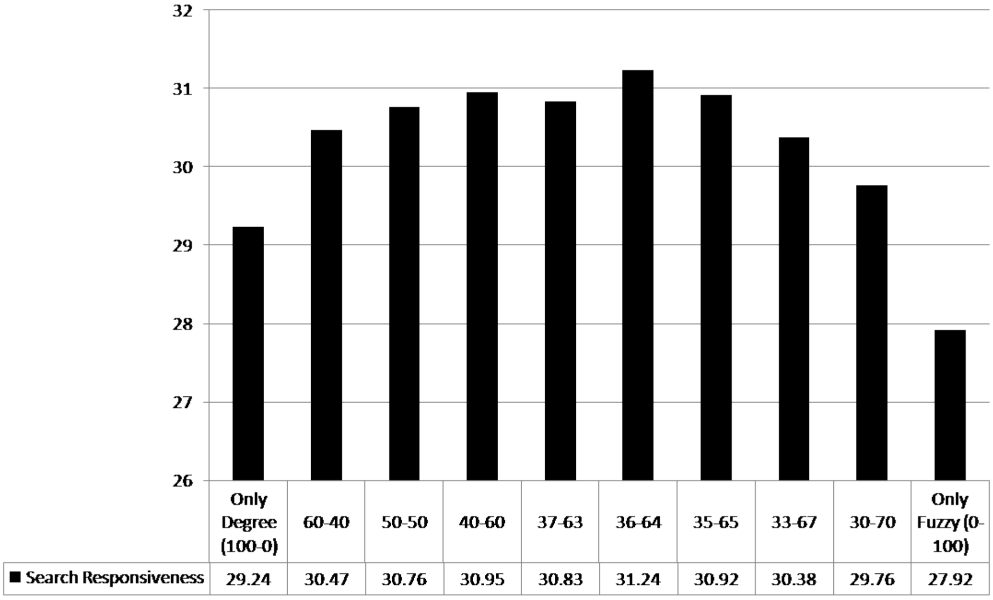
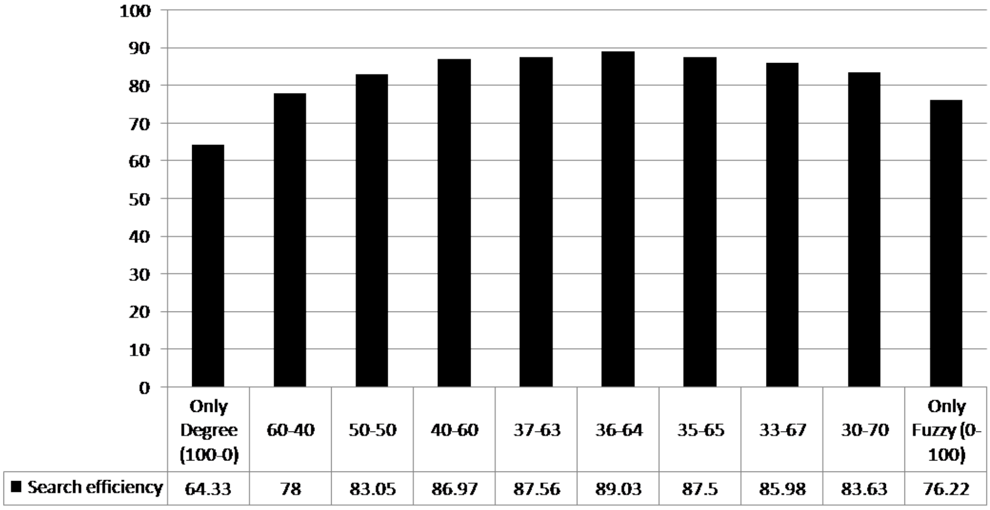


Figure 3.19: Search responsiveness of the hybrid approach - 60-40 indicates 60% weight given to maximum-node-degree and 40% weight given to fuzzy-probability

3.2.5 Conclusion

We have proposed a new approach for neighbour subset selection. Many of the previous approaches are based on the heuristics which don't consider the incoming query expectations. Cosine similarity approach considers the incoming query expectations but considers only the past answering profile of the neighbours to direct queries. Maximum-



	Only Degree (100-0)	60-40	50-50	40-60	37-63	36-64	35-65	33-67	30-70	Only Fuzzy (0-100)
■ Search efficiency	64.33	78	83.05	86.97	87.56	89.03	87.5	85.98	83.63	76.22

Figure 3.20: Search efficiencies of the hybrid approach - 60-40 indicates 60% weight given to maximum-node-degree and 40% weight given to fuzzy-probability

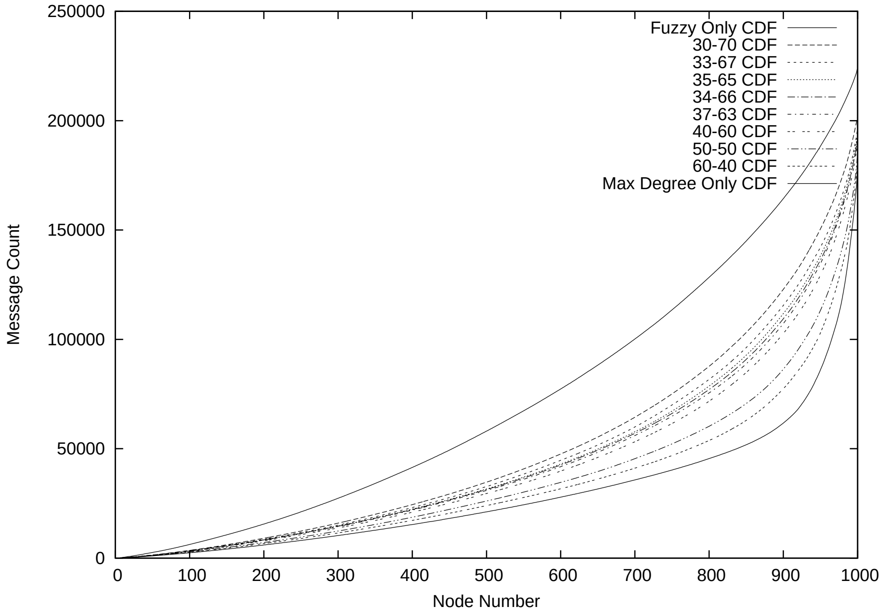


Figure 3.21: CDF of messages per node for the hybrid approach - CDF means Cumulative Distribution Function

node-degree approach is very good at systematically propagating the queries but lacks in success rate and has huge loss of messages. Fuzzy approach guides the queries to those neighbours which have the similar tendencies in content sharing. Simulations show that fuzzy approach has increased the search efficiency by 18.5%. Also by combining the fuzzy approach with maximum-node-degree approach, the best of both are combined with 38% increase in search efficiency.

3.3 Improving Efficiency by Indexing

In the previous section 3.2, search problem is addressed by modifying routing procedure. It reduces traffic but yet keeping the search results at par. In this section, the search problem is approached through indexing. No information retrieval system can function efficiently without indexing. Indexing works through indirection. Pointers to objects are collected and placed in a particular order to create an index. Once the index is created, the lookup for any data item happens on the index first. Then the information of the location of the data item is known. Accessing the data item takes only $O(1)$ time. In centralized systems, index is created and maintained at a single location for all data items.

In a fully decentralized system, data is spread across the nodes in the network. Each node has information only about its own data items. In such a situation, pointers or indexes to data items that belong to other nodes can be created. These indexes will give information about where does the data item is located. But the question is how to create such indexes. Indexes can include all data items or few data items in the network. This decision effects the number of messages required to create the index. To create indexes, nodes need to exchange special messages telling about each other about the data items. In a very large network, the number of messages also increase exponentially. Also maintaining the consistency of the indexes becomes a problem here.

In unstructured P2P networks, the lookup for an object involves forwarding the queries to all neighbours. This generates huge message traffic in the network. As the queries are forwarded, the traffic exponentially multiplies at each hop. This degrades the network performance and thus limits the scalability in terms of number of queries. As the number of queries increase in the network, the whole network bandwidth is consumed by these query messages. This burdens the peers with unnecessary duplicate messages and message processing resulting in huge response times to the querying peers. There are ways to utilize the huge traffic generated and reduce further the query explosion. Here a query-based index propagation approach for disseminating the content indices through the query traffic is presented. Every query carries a summarized index of contents of the peers that are on its way. At every node, the indices are copied to local cache and the local indices are updated in the query message. The nodes can lookup the local index cache and directly contact the node for the object. If the index record is not found, the search falls back on the default lookup algorithm. As time passes, the query traffic reduces and reaches a steady state. In this work, we focus on how to best utilise the query traffic to disseminate the indexes, and evaluate different methods of doing it.

3.3.1 Indexing in Peer-to-Peer Search

Search in unstructured peer-to-peer networks is carried out with the help of several nodes. In this situation, indexing means storing the address of the node where the data item is located. Creation of index involves exchange of messages among the nodes. The aim of indexing in such a situation would be to reduce the number of hops taken to find the object and at the same time balancing with index maintenance cost. Index maintenance cost is measured in terms of number of messages required to create and update the index. [Zhang & Hu 2007] proposed a taxonomy for indexing schemes in peer-to-peer systems. They have classified the schemes in terms of scaling of number of nodes and number of objects. This classification is shown in table 3.9.

Table 3.9: Classification of pee-to-peer indexing schemes - columns indicate the scope of nodes and rows indicate the scope of the objects

	Nodes in Neighbourhood	**All Nodes at Global-scale**
Partial Data Items	Routing indexes	Assisted search
Total Data Items	Floating Indexes (our approach), Probabilistic location routing using attenuated bloom filters	pSearch , eSearch

Routing indexes [Crespo & Garcia-Molina 2002] method indexes subject areas and the number of items available for that subject in a neighbour. Scope of indexing is limited to only for local neighbour hood. In attenuated bloom filter based approach [Rhea & Kubiatowicz 2002], all items are indexed but only the indexes of local neighbourhood are stored at one place. Our approach also approximately fits into this category. Assisted search [Zhang & Hu 2007], pSearch [Tang *et al.* 2003], and eSearch [Tang & Dwarkadas 2004a] are discussed in chapter 2. In the proposed approach, all items are indexed but the indexes are distributed through queries. Therefore indexes may reach every node in the network or may reach only a few. It depends on several factors. At least it can be confidently said that every node will have indexes of its neighbourhood.

3.3.2 Proposed Indexing Scheme

The approach encourages the idea that instead of using separate messages, index information is disseminated by the query traffic. There is always a trade-off between the scope of the index and maintenance cost. In one extreme every node can keep index of all the objects available in the network. That will surely make all the queries one-hop (zero hops for lookup and one hop for getting the object). But the cost associated with maintaining such an index prohibits that approach. On the other extreme searching loses its grace without indexing. Indexing is the most important tool for searching [Baeza-Yates & Ribeiro-Neto 1999].

Our approach is based on day-to-day observation that information about festivals, events and government policies are propagated the posters on the public transportation buses, trains etc. The vehicles communicate the information to the people wherever they go. There is additional cost for transport. We augment that idea to suit the peer-to-peer networks. Each query can be seen as a vehicle carrying the index information of all the nodes it has passed through. The nodes receiving the query copy all the index information. Before forwarding the query it adds its indices to the query if there is free space in the query. This way whoever the query reaches gets benefited. When the node needs to find the object, it looks up its index cache and if there is a match it will immediately contact the peer.

Our approach is fundamentally based on the observation that query traffic of public file sharing peer-to-peer networks occupies most of the internet traffic. In [Saroiu *et al.* 2002a], it is observed that p2p network traffic occupies the 40% of total internet traffic. In [Gummadi *et al.* 2003], it is reported that there are 562 transactions per second happening in Kazaa [kaz 2001] network. This clearly gives the extent of queries in p2p systems. Not only successful queries but even unsuccessful queries can be used as vehicles for carrying index information. Not only queries but even the query replies and download requests can be used for this purpose. This provides us with a large percentage of traffic for conveying index information.

3.3.2.1 Index Creation

A query passes through many nodes and if every node adds its index to the query, query will increase in size and will consume huge bandwidth. To address this issue we use Bloom Filers to store and send only the summarized index for specified number of files in one node. Bloom Filters [Bloom 1970] are widely used to represent elements of a set using a bit set hashed through a set of hash functions. The filter may report an element is present even though it is not present in the set. This is known as false positive. But the filter never reports absence of an element when it is present in the set. The false positive probability (fpp) of representing set of elements of size n depends on size (in number of bits) m of Bloom Filter and size k of the set of hash functions. From equation 3.24, it can be noted that fpp reduces exponentially as n is reduced, and as k and m are increased. Typically m and fpp are computed as follows.

$$m = \frac{k * n}{\ln 2} \tag{3.23}$$

$$fpp = (0.6185) \frac{m}{n} \tag{3.24}$$

Now let us apply Bloom Filters to Gnutella protocol. Here we have a set of constraints. First, query message size is limited. Maximum query size as per Gnutella protocol 0.6 [gnu 2002], is 4 KB. The number of files shared by users varies from zero to thousands. So to have one bloom filter for all files in a node is not feasible. Second, Bloom Filters don't have an option for deleting elements. It can only add new elements into the filter. So nodes create a set of Bloom Filters for the objects shared for the network. Nodes exchange Bloom Filters not a part of it. That is if there is an update in a Bloom Filter, entire Bloom Filter is exchanged with neighbour nodes. So it requires identifying each Bloom Filter uniquely across the network so that it is possible to replace old Bloom Filter with new one.

As observed in [Saroiu *et al.* 2002b], in a p2p file sharing network, 75% of nodes share 100 or less number of files. So it will be appropriate to group 100 files of a node and represent their file names in one Bloom Filter. Nodes having more than 100 files will have more than one Bloom Filter to convey their indices. By fixing size of n at 100, we have various alternatives in choosing the sizes of m and k. [Fan *et al.* 2000] documents the fpp values for optimised combinations of $\frac{m}{n}$ and k. As k increases fpp falls exponentially. The choice of k effects the bandwidth consumption, and the time required to disseminate the indices.

For supporting efficient dissemination and updating of indices across the network without extra cost we propose the following structure called Query Index Record (QIR). QIR record consists of the following fields.

- N: node identity such as IP address
- B: the bandwidth (mbps) of the connection available at node N
- R, S, D_c: set of Bloom Filter Record (BFR) at N
    - R: record number unique for N
    - S: m bit vector
    - D_c: date on which Bloom filter is created by N. It is measured as seconds that have elapsed since the starting of the network. This date is used to compare the freshness of the BFR.

A BFR is uniquely identified in the network by the combination of N and R.

A query carries several QIR records as many as it can fit in its message. Let us see an example of estimating the number of indexes carried by a query. As observed in [Saroiu *et al.* 2002b], a node is sharing maximum of 8000 files. Each BFR can store summary of

100 files as discussed above. The value of R doesn't exceed 100 in worst case thus takes 1 byte space. For $k = 6$, the size of BFR record in bytes is (1, 109, 4) = 114 bytes. The size of QIR record in bytes is (4, 1, size of BFR set). Also each node has a local cache of Node Index Records (NIR). When a query arrives at a node, the QIR is stored at the local node in the form of NIR. Each NIR has the following fields in addition to the fields in QIR.

- D_{lu}: the date on which it was last successfully used. D_{lu} is used for removing the NIR after certain time.
- D_{ll}: the date on which it was loaded on to a query. D_{ll} is used in deciding which indices to be loaded onto the query.

A Gnutella 0.6 Query message takes maximum of 54 bytes assuming that keywords usually take less than 30 characters thus be within maximum of 30 bytes. Maximum message size is 4 KB. Then it is possible to fit in $\frac{4096-54}{119} = 35$ index records in each Query message. The 35 index records convey the locations of 3500 objects in the network. What should be done when there are more than 35 index records to load on to a query is discussed in next section.

So, indexes for data items are created in units of Bloom filters. Each bloom filter is identifiable uniquely in the network.

3.3.2.2 Index Dissemination

In an unstructured peer-to-peer network, queries are propagated either through flooding or through a random-walk or by a hybrid method. Flooding is a breadth-first search method where as random-walk is a depth first search method. In the proposed indexing scheme, the indexes are propagated through queries. Therefore index dissemination is completely dependent on how well queries are propagated. Three parameters that characterize this are:

- **Query density**: This indicates how query firing is spread out in a network. In this work, it is assumed that approximately each node fires the same number of queries.
- **Query rate**: It is the number of queries fired by a node per unit time. If the rate at which queries are fired is γ, then the probability that a node will fire q queries in t

time units is

$$P(q,t) = \frac{(t\gamma)^q}{q!} e^{-n\gamma} \tag{3.25}$$

- **Query forwarding method**: There are two primary methods, i.e. flooding and random-walk. In flooding, a single query is cloned into all neighbours at every hop. But in random walk with k-walkers, at every hop, a query is forwarded to only one neighbour. In flooding query traffic is much higher than that of random walk.

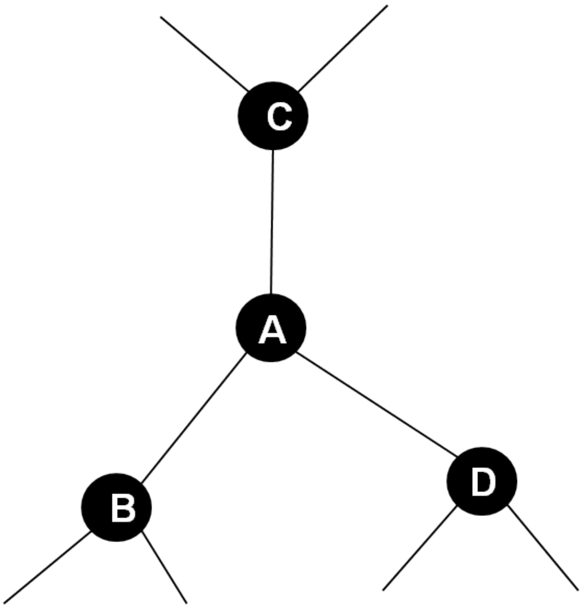


Figure 3.22: A simplified system model of unstructured peer-to-peer network - A has d neighbours. Every node has d neighbours.

Consider a simple model shown in figure 3.22. Assume that for transferring a message or query from one node to another, one unit of time is taken. Every node fires γ queries per unit time. In case of flooding,

- at t_0, every node fires γ queries.
- at t_1, every query advances by one hop. At node A, the number of incoming queries are $d\gamma$.
- at t_2, number of incoming queries are $d\gamma + \sum_{i=0}^{d-1} neighbourCount(i) \times \gamma$.

Steady state is reached when $t = TTL$. At steady state the number of queries (β) entering node A per unit time is

$$\begin{aligned} \beta_A &= d\gamma + d(d-1)\gamma + d(d-1)^2\gamma + \ldots + d(d-1)^{TTL-1}\gamma \\ &= d\gamma\frac{1-(d-1)^{TTL-1}}{2-d} \end{aligned} \tag{3.26}$$

In case of random-walk having k walkers,

- at t_0, every node fires γ queries.
- at t_1, at node A, the number of incoming queries are $\frac{k}{d}\gamma$. $\frac{k}{d}$ is the probability of choosing node A by a neighbour to forward the query.
- at t_2, number of incoming queries are $\frac{k}{d}\gamma + \frac{k}{d-1}\gamma$.
- at t_3, number of incoming queries are $\frac{k}{d}\gamma + \frac{k}{d-1}\gamma + \frac{k}{(d-1)^2}\gamma$.

At steady state ($t = TTL$), the number of queries (β) entering node A per unit time is

$$\begin{aligned} \beta_A &= \frac{k}{d}\gamma + \frac{k}{d-1}\gamma + \frac{k}{(d-1)^2}\gamma + \ldots + \frac{k}{(d-1)^{TTL-1}}\gamma \\ &= k\gamma(1 + \frac{1}{d-1} + \frac{1}{(d-1)^2} + \ldots + \frac{1}{(d-1)^{TTL-1}}) \\ &= k\gamma\frac{1-(\frac{1}{d-1})^{TTL-1}}{1-\frac{1}{d-1}} \\ &\approx k\gamma(\frac{d-1}{d-2}) && \text{for large TTL values} \end{aligned} \tag{3.27}$$

Equations 3.26 and 3.27 estimate the maximum number of queries that enter a node per unit time. Many of the queries might have found a match or they would have become a duplicate message etc. In flooding, the probability of a query reaching the same node which it had reached before is quite high. But in random walk, this is very low.

Index creation in each local node is described in section 3.3.2.1. These indexes are disseminated either through flooding or through random-walk methods. Since the indexes are floated through queries, this indexing scheme is called as 'Floating Indexes(FI)'. There are two ways in which floating indexes can be replicated.

- **Floating indexes Breadth-wise (FIB)**: In this scheme, the query relinquishes all the indexes in the local node and takes in a set of indexes which are given by the local node. In figure 3.23, node A has (x, y, z) indexes stored with it. The incoming query is carrying (a, b, c) indexes. When the query reaches A, all the indexes in the query are copied to cache of A and a set of indexes in A are loaded onto the query based on some selection criteria.

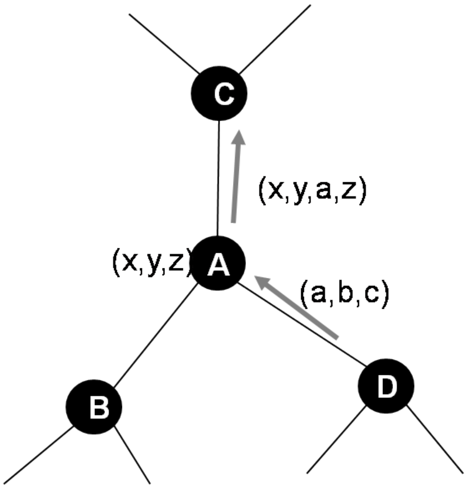


Figure 3.23: Floating indexes Breadth-wise (FIB) scheme - A has (x, y, z) indexes. Incoming query is carrying (a, b, c) indexes

- **Floating indexes Depth-wise (FID)**: In this scheme, the indexes in the query are preserved. In addition, if there is free space, local indexes are added to the query. The original indexes are not relinquished. In figure 3.24, when the query reaches node A with indexes (a, b, c), the indexes are copied to local node and the indexes with A are added to the query if there is space.

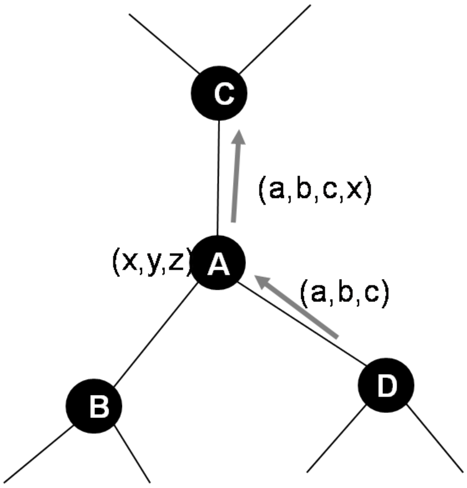


Figure 3.24: Floating indexes Depth-wise (FID) scheme - A has (x, y, z) indexes. Incoming query is carrying (a, b, c) indexes

In FIB scheme, an index at node A is spread in proportion to the number of queries processed by node A. In FID scheme, the indexes are spread in proportion to the number of queries fired by the node. In FIB, incidents of duplicate indexes reaching the same where they are already present are far less than in FID where the query carries the index of the source node intact. In FIB scheme, even if the node itself doesn't have too many queries to fire, still its local indexes are propagated equally. In FID scheme, the node which fires a search query propagates its indexes to the depth of TTL. If a node doesn't have too many search queries but has many files to share, its indexes are not well spread. In flooding, due to vast coverage of the queries, overall both the schemes work the same way. But in random walk, FID scheme spreads the indexes in more uniform way than in FIB scheme due to random selection of neighbours to forward the queries. For propagating updates or the changed indexes, in FIB it is quicker because there are $k\gamma(\frac{d-1}{d-2})t$ queries in t time units where as FID has only γt queries in the strict sense.

3.3.2.3 Search Procedure

Node Join: When a node joins the network, during the initial handshake with neighbours, it collects NIR from each neighbour and updates its local index cache. Acquiring NIR from queries alone would take enormous time given the fact that nodes would use indexes to lookup the objects instead of queries. Also, node creates NIR records for the files it wants to share.

Search: When a node wants to lookup for set of keywords W, it first looks up its own files and index as shown in the search algorithm in 3.25. If successful, the output of this algorithm is a set of hosts H which may have the file. Out of set of hosts H, $choseHostWithMaxBw()$ procedure chooses the host h with maximum bandwidth. If there are no matches, it will resort to usual lookup method i.e, flooding or random walk. It creates a new query. The query q is populated with the NIR. In case the node has more indices than it could fit in the query, it will choose the indices based on D_{II} as shown in 3.27. The indices are chosen in a round-robin fashion ensuring that all the indices are propagated. The $sort()$ method sorts the index records on D_{II} in ascending order. Similarly when a node receives the query, procedure given in 3.26 is executed. The node will scan the incoming query q for the QIR records and either replaces or adds them to local cache. The criterion for either replacing or adding is shown in 3.28. After updating the local index cache, it will search files and the local index cache. If it finds a match or matches, it will make a reply message and includes the addresses and the bandwidth information of all the nodes it has found to have the desired object. But it doesn't confirm whether the object really exists in the network. So the result may include the false positives also. If it doesn't find a match, it will forward the query to its neighbours.

```
begin
    NodeIdentity: N; Bandwidth: B; Result: $R$;
    Host: $H$;
    IndexCache: $I$;
    Files: $F$;
    Neighbours: $P$;
    foreach file f in F do
        if match(f, W) = true then
            $R \leftarrow R \cup f$;
        end
    end
    if $R = \phi$ then
        foreach index record i in I do
            if BloomFilterMatch(i, W) then
                $H \leftarrow H \cup (N, B)_i$;
            end
        end
    else
        return R;
    end
    if $H = \phi$ then
        $q \leftarrow makeQuery(W)$;
        $q \leftarrow loadIndex(q, I)$;
        foreach neighbour n in P do
            forward q to n;
        end
    else
        $h \leftarrow choseHostWithMaxBw(H)$;
        $requestDownload(h, W)$;
    end
end
```

Figure 3.25: Search procedure at source node: *Search(Keywords W)*

```
begin
    NodeIdentity: N; Bandwidth: B; Result: $R$;
    Host: $H$;
    IndexCache: $I$;
    Files: $F$;
    Neighbours: $P$;
    Guidcache: $C$;
    $updateIndexinNode(q, I)$;
    if guid(q) in C then
        return
    end
    foreach file f in F do
        if $match(keyword(f), keyword(q)) = true$ then
            R ← R ∪ f
        end
    end
    if $R = \phi$ then
        foreach index record i in I do
            if $BloomFilterMatch(i, W)$ then
                $H \leftarrow H \cup (N, B)_i$;
            end
        end
    else
        return R;
    end
    if $H = \phi$ then
        $q \leftarrow loadIndex(q, I)$;
        select a neighbour n from P at random;
        forward q to n;
    else
        send H ∪ R to sender;
    end
end
```

Figure 3.26: Query processing at a node: *ReceiveQuery(Query q, Node sender)*

```
begin
    Int: $QSize$;
    $I \leftarrow sort(I, ''D_{ll}'')$;
    $q \leftarrow clearIndexes(q)$;
    $QSize \leftarrow 0$;
    foreach index i in I do
        load $(N, B, R, S, D_c)_i$ onto q;
        set $D_{ll}$ to current time;
        $QSize \leftarrow QSize + QIR\_SIZE$;
        if $QSize + QIR\_SIZE > QUERY\_MSG\_SIZE$ then
            return;
        end
    end
end
```

Figure 3.27: Procedure for loading index onto a query: *loadIndex(Query q, IndexCache I)*

```
begin
    boolean: $found$;
    foreach index record r in R do
        $found \leftarrow false$;
        foreach index record i in I do
            if $N_i = N_r$ and $R_i = R_r$ then
                $found \leftarrow true$;
                if $D_c(i) < D_c(r)$ then
                    $i \leftarrow r$;
                end
            end
        end
    end
    if not found then
        $I \leftarrow I \cup r$;
    end
end
```

Figure 3.28: Procedure for copying indexes from query to node: *updateIndexinNode(Query q, IndexCache I)*

3.3.3 Experiment Setup

The simulator model is discussed in section 3.2.3.1. We used a 1000 node random graph generated by GT-ITM [Zegura *et al.* 1996] with average degree of 9.3. We used a pure P2P model where there is no distinction among the peers. All peers have equal functionalities. There are 6000 distinct objects in the network with the 1st object being the most popular and 6000^{th} object being the least popular. The objects and queries are distributed according to the observations made in [Chu *et al.* 2002]. The replica distribution of these objects is done according to the Zipfian distribution with parameter $\alpha = 0.82$. The replicas are placed at randomly selected nodes. Queries for these objects also follow the Zipfian distribution ($\alpha = 0.82$) i.e. the popular objects receive more queries than that of less popular objects. Total number of objects in the network including the replicas are 28137. Average number of objects per node are 36.5. There are 10000 queries, each query being fired from a randomly chosen node. The lapse between two consequent queries is approximately 100 milli seconds. The node distribution is kept the same for all simulations. But the file distribution, and query firing order are kept in two different sets. Results of set1 are shown here. Results from set2 are similar to that of set1.

Churn: Churn refers to the events of nodes joining and leaving the system. The rate of these events is high in peer-to-peer file sharing networks. To model the churn scenario, after every 500 queries, 30 nodes are removed from the network and 30 nodes are added. The new nodes start functioning without any prior knowledge. A set of new objects are replicated onto these new nodes according to Zipfian distribution. For simulating the search method flooding, the TTL is set to 4 and for k-random walk 4 walkers are used and TTL for each is set to 1024. The parameters for Bloom Filter are number of files $m/n = 32$, $k = 8$. With these parameters there is false positive probability of $5.73 \times e^{-06}$ [Fan *et al.* 2000]. Maximum query size is 4096 bytes.

3.3.4 Result Analysis

The purpose of this simulation is to verify the effectiveness of floating indexing scheme and compare it with the attenuated bloom filters (ABF) or probabilistic routing scheme proposed in [Rhea & Kubiatowicz 2002]. Simulations are carried out for various combinations:

- Flooding TTL=4
- Random-walk with 4 walkers
- Floating indexes with flooding with breadth-wise spread
- Floating indexes with random-walk with breadth-wise spread (FIB)
- Floating indexes with random-walk with depth-wise spread (FID)
- Floating indexes with fuzzy-walk with breadth-wise spread (fuzzyFIB). This is the integration with the solution proposed in the section 3.2.
- Attenuated Bloom Filters (ABF) with levels 2 and 3

3.3.4.1 Flooding

Table 3.10 displays the summary findings for flooding search method for various TTL values.

Table 3.10: Search performance metrics computed for Flooding approach - these statics are obtained from simulations done in the same environment

	TTL=1	**TTL=2**	**TTL=3**	**TTL=4**
Number of nodes	1000	1000	1000	1000
Total query hits	2672	9216	33818	56647
Successful queries	951	3331	8612	9908
Total messages	112367	1012711	7415698	23280193
Total queries fired	10001	10001	10001	10001
Average Hops from HitNode	0.9	1.68	2.65	3.45
Avg Query efficiency	0.24	0.09	0.05	0.02
Search Responsiveness	10.57	19.84	32.49	28.68
Search efficiency	2.54	1.79	1.62	0.57

Query efficiency, search responsiveness and search efficiency are all computed as per equations 3.6, 3.5, and 3.8. Flooding has the nature of quick responses and huge overhead on the network. This can be understood from this table. There is an exponential increase in number of messages as the TTL value is increased. Search efficiency decreases drastically due to this.

3.3.4.2 Random-walk

Table 3.11 shows the summary findings for random-walk search method.

Table 3.11: Search performance metrics computed for Randomwalk approaches - w refers to number of walkers. These statistics are obtained from simulations done in the same environment

	w=1	w=2	w=3	w=4	w=5	w=6	w=7	w=8	w=9	w=10
Number of nodes	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
Total query hits	2833	3676	4422	5074	5745	6303	6661	7239	7310	7674
Successful queries	2343	2711	2984	3269	3474	3669	3680	3923	3876	4026
Total messages	623554	779166	906705	1015052	1114270	1204466	1268964	1332066	1375369	1412537
Total queries fired	10001	10001	10001	10001	10001	10001	10001	10001	10001	10001
Average Hops from HitNode	13.93	11.97	10.59	9.65	8.78	8.40	7.79	7.56	7.23	7.33
Avg Query efficiency	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
Search Responsiveness	1.68	2.26	2.82	3.39	3.95	4.37	4.73	5.19	5.36	5.49
Search efficiency	0.08	0.11	0.14	0.17	0.2	0.22	0.24	0.26	0.27	0.27

When compared with flooding, random-walk has higher hop-counts and thus has lower response values. Random-walks have very low success rate. As the number of walkers are increased, there is decrease in hop-count and increase in success ration. This contributes to increase in efficiency. As number of walkers are increased to a value around the average node degree, i.e., 9.33, efficiency reaches maximum value.

3.3.4.3 Floating Indexes Breadth-wise through Random-walk (FIBRW)

Table 3.12 shows the results of simulations carried out using floating indexes with breadth-wise spread (FIB) through random-walk. Success ratio is more than 99%. The average hop-count or pathlength to the object is less than 2 hops much less than that of random-walk which is around 8 hops. This is very interesting.

Table 3.12: Search performance metrics computed for Floating Indexes with Breadthwise spread (FIB) - **w** refers to number of walkers. These statistics are obtained from simulations done in the same environment

	w=1	**w=2**	**w=3**	**w=4**	**w=5**	**w=6**	**w=7**	**w=8**	**w=9**
Number of nodes	1000	1000	1000	1000	1000	1000	1000	1000	1000
Total query hits	15507	17554	19205	20728	21693	22470	23514	23526	24097
Successful queries	9957	9973	9971	9976	9983	9982	9979	9984	9988
Total messages	65382	73905	79864	84060	89934	92927	96460	96317	99350
Total queries fired	10001	10001	10001	10001	10001	10001	10001	10001	10001
Average Hops from HitNode	1.59	1.64	1.64	1.62	1.67	1.65	1.64	1.62	1.66
Avg Query efficiency	2.37	2.37	2.4	2.47	2.41	2.42	2.44	2.44	2.43
Search Responsiveness	62.78	60.63	60.96	61.63	59.9	60.46	60.99	61.61	60.31
FI Breadhwise Search Eff.	148.79	143.69	146.3	152.23	144.36	146.31	148.82	150.33	146.55

3.3.4.4 Floating Indexes Depth-wise through Random-walk (FIDRW)

Table 3.13 shows the results of propagating indexes depth-wise with random walkers.

Table 3.13: Search performance metrics computed for floating indexes depth-wise (FID) with randomwalkers - w refers to number of random walkers. These statistics are obtained from simulations done in the same environment

	FID RW w=4	FID RW w=8	FID RW w=9
Number of nodes	1000	1000	1000
Total query hits	18241	21877	21354
Successful queries	9707	9750	9730
Total messages	148934	181714	179300
Total queries fired	10001	10001	10001
Average Hops from HitNode	2.35	2.37	2.28
Avg Query efficiency	1.22	1.2	1.19
Search Responsiveness	41.31	41.16	42.58
FI Depthwise Search Eff.	50.4	49.39	50.67

Success ratio is low as around 300 queries are unsuccessful. Its hop count is higher and search efficiencies are three times below when compared with breadth-wise spread.

3.3.4.5 Floating Indexes Breadth-wise through Fuzzy-walker (FIBFuzzy)

The table 3.14 shows the results of simulations on the approach that combines floating indexes and fuzzy walkers approach. Fuzzy walkers approach was discussed in section 3.2. Here the walkers are guided by the fuzzy probability values of neighbours instead of random selection. This has both advantages and disadvantages as we see ahead.

The fuzzy results are slightly better than that of FIBRW. Here the pathlength to hit node is smaller and has lower success ratio. The reduction in pathlength is due to fuzzy probabilities.

Table 3.14: Search performance metrics computed for Floating Indexes with Breadthwise spread (FIB) using fuzzy walkers - w refers to number of walkers. These statistics are obtained from simulations done in the same environment

	Fuzzy FIB w=1	Fuzzy FIB w=2	Fuzzy FIB w=3	Fuzzy FIB w=4	Fuzzy FIB w=5	Fuzzy FIB w=6
Number of nodes	1000	1000	1000	1000	1000	1000
Total query hits	22288	27106	30623	33226	34630	34772
Successful queries	9775	9818	9834	9869	9893	9920
Total messages	85705	101226	112815	122427	128008	130906
Total queries fired	10001	10001	10001	10001	10001	10001
Average Hops from HitNode	1.50	1.51	1.50	1.53	1.54	1.57
Avg Query efficiency	2.6	2.68	2.71	2.71	2.71	2.66
Search Responsiveness	65.15	65.05	65.41	64.43	64.33	63.22
Fuzzy FI Search eff.	169.39	174.33	177.26	174.61	174.33	168.17

3.3.4.6 Attenuated BloomFilters(ABF)

Table 3.15 shows the results of simulations on probabilistic routing algorithm using attenuated bloom filters with 3 levels.

Table 3.15: Search performance metrics computed for attenuated bloom filters (ABFL3) - w refers to number of walkers. The depth of bloom filter array is 3. These statistics are obtained from simulations done in the same environment

	w=1	w=2	w=3	w=4	w=5
Number of nodes	1000	1000	1000	1000	1000
Total query hits	18534	21179	23244	24822	26103
Successful queries	9963	9979	9988	9993	9996
Total messages	2636774	2697374	2752060	2803030	2848957
Total queries fired	10001	10001	10001	10001	10001
Average Hops from HitNode	4.30	4.09	3.91	3.77	3.67
Avg Query efficiency	0.07	0.08	0.08	0.09	0.09
Search Responsiveness	23.18	24.41	25.55	26.49	27.27
Search efficiency	1.62	1.95	2.04	2.38	2.45

It has high success ratio but huge message overhead. These messages are predominantly due to index building and propagation.

3.3.4.7 Floating Indexes Breadth-wise through Flooding (FIBFL)

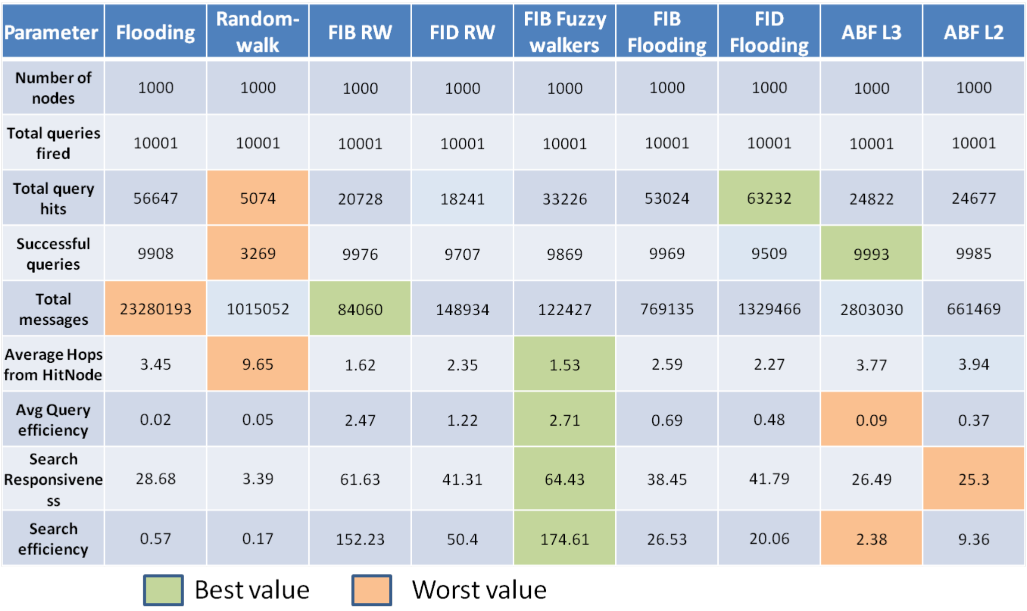
Table 3.16 shows the results of simulations on Floating indexes breadth-wise (FIB) approach with flooding as the carrier. The results are better than basic methods like flooding and randomwalk but far below when compared with FIB with randomwalk as the carrier. Major difference is in number of messages generated.

Table 3.16: Search performance metrics computed for floating indexes with flooding - TTL refers to hop limit for each query. These statistics are obtained from simulations done in the same environment

	TTL=2	**TTL=4**
Number of nodes	1000	1000
Total query hits	32565	53024
Successful queries	9447	9969
Total messages	210659	769135
Total queries fired	10001	10001
Average Hops from HitNode	1.38	2.59
Avg Query efficiency	1.55	0.69
Search Responsiveness	68.38	38.45
Search efficiency	105.99	26.53

3.3.4.8 Results Summary

The summary of the statistics are shown in figure 3.29. Figure 3.30 shows the comparison between basic search methods such as flooding and random-walk and floating indexes scheme and probabilistic routing algorithm schemes. There is a huge improvement in floating indexes schemes.



Parameter	Flooding	Random-walk	FIB RW	FID RW	FIB Fuzzy walkers	FIB Flooding	FID Flooding	ABF L3	ABF L2
Number of nodes	1000	1000	1000	1000	1000	1000	1000	1000	1000
Total queries fired	10001	10001	10001	10001	10001	10001	10001	10001	10001
Total query hits	56647	5074	20728	18241	33226	53024	63232	24822	24677
Successful queries	9908	3269	9976	9707	9869	9969	9509	9993	9985
Total messages	23280193	1015052	84060	148934	122427	769135	1329466	2803030	661469
Average Hops from HitNode	3.45	9.65	1.62	2.35	1.53	2.59	2.27	3.77	3.94
Avg Query efficiency	0.02	0.05	2.47	1.22	2.71	0.69	0.48	0.09	0.37
Search Responsiveness	28.68	3.39	61.63	41.31	64.43	38.45	41.79	26.49	25.3
Search efficiency	0.57	0.17	152.23	50.4	174.61	26.53	20.06	2.38	9.36

🟩 Best value 🟧 Worst value

Figure 3.29: Summary statistics of index dissemination methods - TTL for methods with Floodng as carrier is 4 and number of walkers are 4. FIB refers to Floating Indexes with breadth-wise spread, FID refers to Floating Indexes with depth-wise spread, ABF refers to attenuated bloom filters

- **Floating Indexes Vs Flooding & Random-walk:** When we compare Floating Indexes approach with basic methods flooding and random-walk there is a clear improvement in search efficiency. The efficiency improvement is prominently due to higher success rate, lower pathlenghts and reduction in message traffic. The efficiencies of flooding and random-walk approaches are only 0.57 and 0.17. Search efficiency of all 'Floating Indexes' approaches is above 20.

- **Floating Indexes Vs Attenuated BloomFilters:** Certainly attenuated bloom filters approach reduces pathlength to 3.77 and increases success rate to 99.99%. But its only drawback is it requires huge number of messages to build the BloomFilters. Since there are no extra messages required for index propagation in Floating Indexes approach, there is enormous improvement search efficiency.

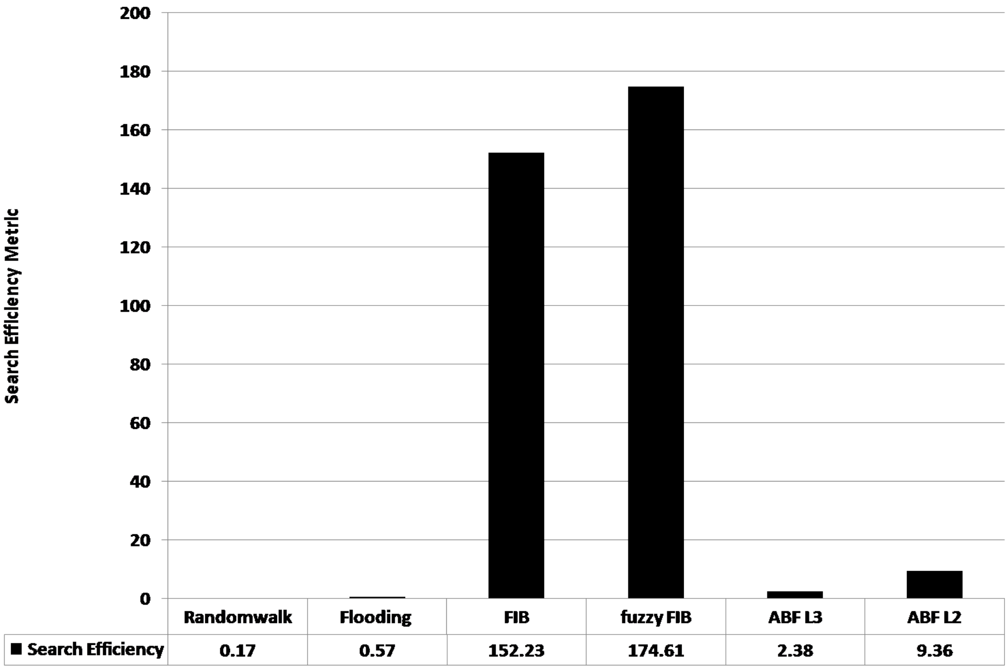
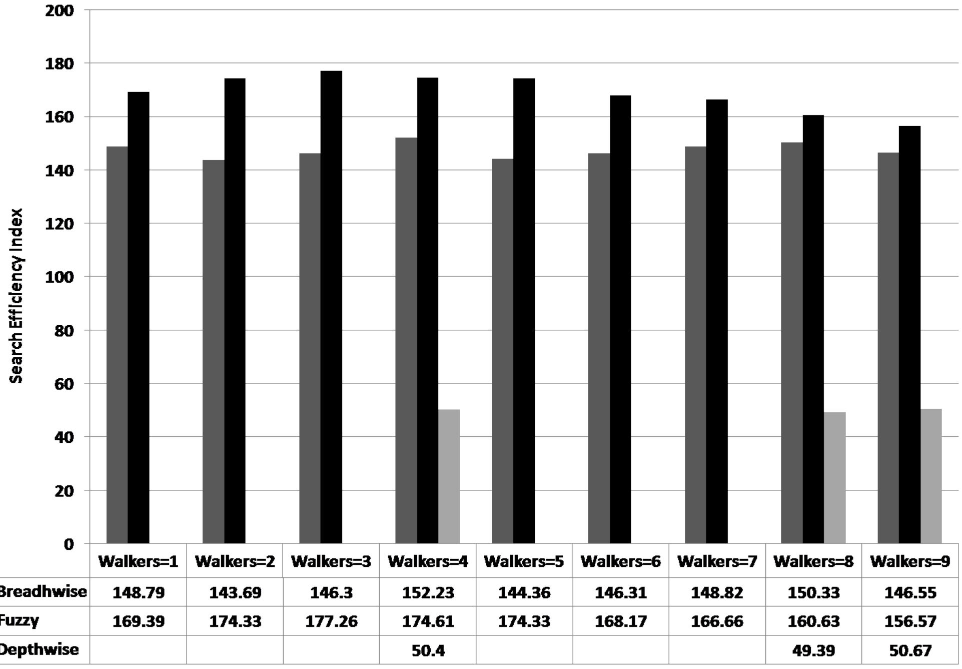


Figure 3.30: Search efficiency metric comparison of Floating Indexes scheme with other methods - FIB refers to Floating Indexes with breadth-wise spread, ABF refers to attenuated bloom filters

- **Carrier methods:** Major difference is that the number messages generated in flooding is high compared to that of random-walk methods. In flooding the majority of messages end up as duplicate messages. Duplicate messages use network resources but don't add indexes to the node because that query is already processed by the node. In the figure it can be noticed that in breadth-wise spreading, messages generated by flooding are 9 times more than that of random-walk, though results are similar.

- **Dissemination Schemes:** Among the breadth-wise and depth-wise schemes, breadth-wise scheme has better results. It is observed that breadth-wise spreading of indices is giving three times better results than depth-wise spreading of indices. From the figure 3.29, depth-wise scheme either through flooding or random-walk, has very low success rate, huge number of messages and very few query hits per query. This is because depth-wise spreading is biased towards spreading indexes of a few nodes all over the network. This is discussed further in the coming section.

- **Effect of number of walkers:** Figure 3.31 shows the search efficiencies with increased number of walkers. The performance remains roughly consistent irrespective of number of walkers.



	Walkers=1	Walkers=2	Walkers=3	Walkers=4	Walkers=5	Walkers=6	Walkers=7	Walkers=8	Walkers=9
■ FI Breadhwise	148.79	143.69	146.3	152.23	144.36	146.31	148.82	150.33	146.55
■ FI Fuzzy	169.39	174.33	177.26	174.61	174.33	168.17	166.66	160.63	156.57
■ FI Depthwise				50.4				49.39	50.67

Figure 3.31: Search efficiency metric comparison among floating indexes schemes with varied number of walkers - FI refers to Floating Indexes

3.3.4.9 Effect of Index Dissemination on Message Traffic & Path-lengths

Figure 3.32 depicts the rate of rise in messages as the nodes fire queries. In pure randomwalk method, the messages increase at a constant rate per query. In floating indexes breadth-wise scheme with flooding as the carrier, initially it is exponential rise in the messages. But as the indexes spread, the queries are answered in nearby neighbourhoods. Therefore we see the slope of the curve reduces and becomes almost flat towards the end. This indicates that at steady state, most of the queries are answered with the foreign indexes available locally. Also we see that there are many spikes in the curve whose frequency reduces as the curve grows. The steeps indicate that certain queries could not be answered within the neighbourhood and that led to sudden increase in the number of messages. In attenuated bloom filters based method (ABF), the slope is almost constant but much less than the pure randomwalk method. That means in attenuated bloom filters based routing method, there are constant number of messages being generated per each query. The three curves at the bottom portion of the graph correspond to three schemes in floating indexes with walkers as the carrier. Among them FIB with random walkers has the lowest slope. Next lowest slope is for FIB with fuzzy walkers and the next is for FI depth-wise dissemination. In fuzzy walkers, at every hop the walkers are directed to a neighbour which has the highest probability value. In this case there is high chance of encountering the same neighbour for most of the queries thus reducing the spreading of the indexes. In FI depth-wise dissemination, the query carries the indexes loaded by first few nodes in the query into the depth of the network. Since walkers coverage of network is very less, the spread is not extensive.

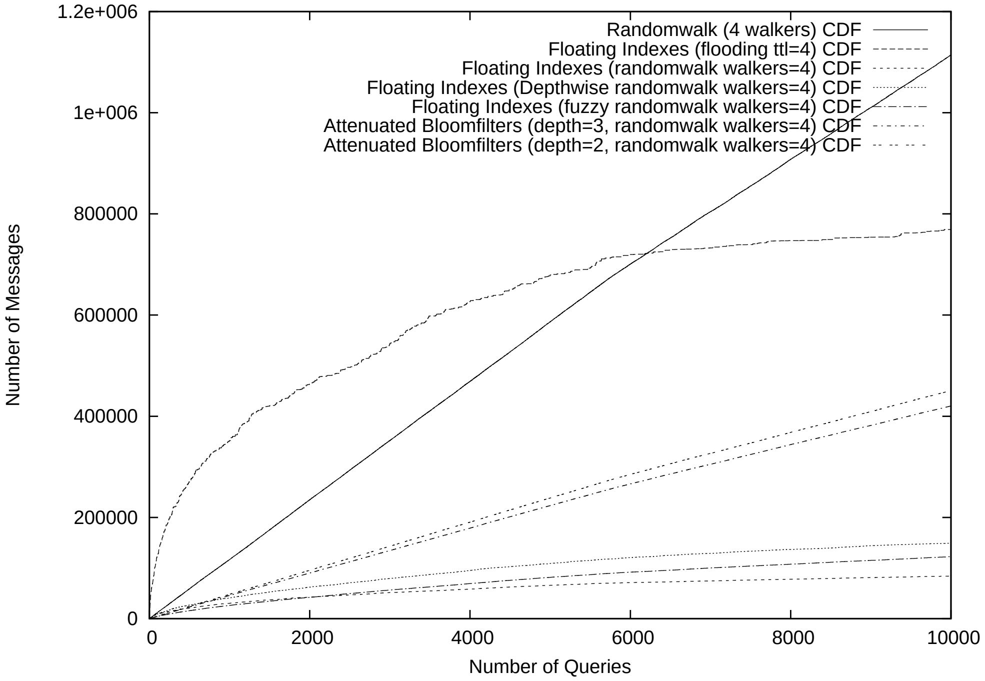


Figure 3.32: Message generation CDF for Floating Indexes and other methods - A point on CDF curve indicates the total number of messages generated so far for the corresponding number of queries on x-axis

Figure 3.33 depicts the pathlengths of query hits. A pathlength is defined as the number of hops between the source node and the queryhit node. Queryhit node is where the query has found a match. In this figure, pure random walk has the maximum slope. That means every query has got the same pathlength and this pathlength is higher than that of other methods. The gap between pure randomwalk and pure flooding increases as the number of queries increase. This indicates that the pathlengths in randomwalk are more than that of flooding. It is observed that Floating indexes have far less growth in pathlength. FI with randomwalk takes time to settle down where as FI with flooding settles down at an early time. In FI due to the spread of indexes, pathelength reduces to zero in most of the queries. Although FI with flooding is offering better pathlengths, but it has the negative point of generating more messages than that of FI with randomwalk. In figure 3.34, the pathlength rate of increase for attenuated bloom filters is much higher compared to schemes of Floating Indexes. Among the floating indexes schemes, Floating Indexes with random walkers has the best result. It has the least growth in pathlengths. It indicates that pathlength per query is very small or likely zero.

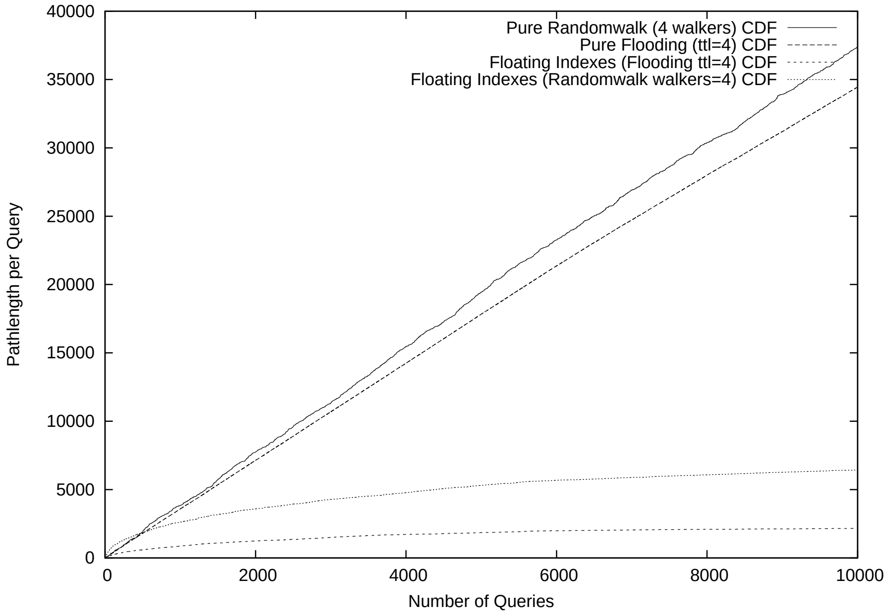


Figure 3.33: Pathlength CDF for Floating Indexes and other basic methods - A point on CDF curve indicates the sum of pathlengths of queryhits received so far for the corresponding number of queries on x-axis

3.3.4.10 Utilization of Query Traffic for Index Dissemination

Figure 3.35 shows that FID spreads the indexes evenly across the network i.e. almost all nodes have acquired equal number of indexes. FIB has bigger variations. But looking at characteristics of indexes spread, in FID, indexes of 50% nodes are spread across more than 80% of the network and indexes of 30% nodes are spread only in less than 20% of

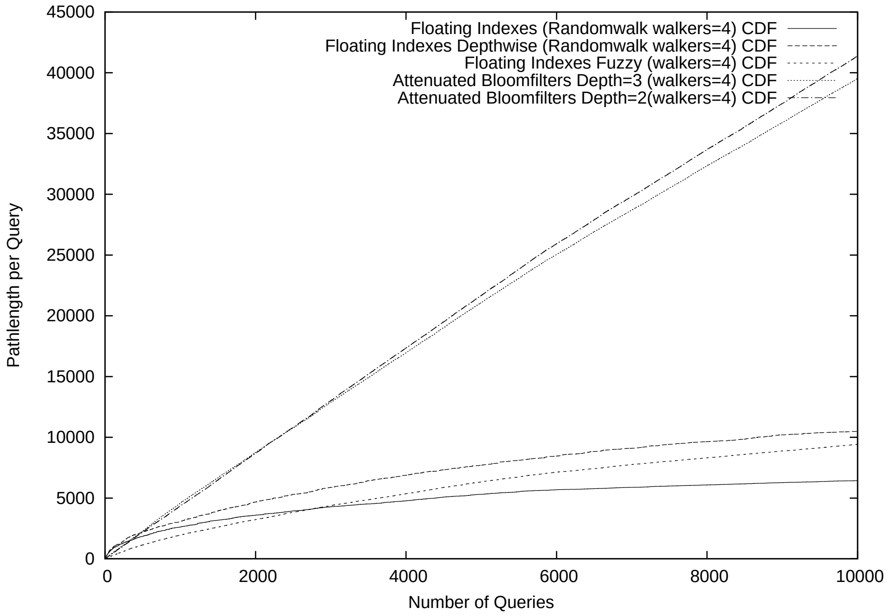


Figure 3.34: Pathlength CDF for different Floating Indexes schemes and Attenuated Bloom Filters - A point on CDF curve indicates the sum of pathlengths of queryhits received so far for the corresponding number of queries on x-axis

the network. In FID, indexes of 6.9% nodes are not distributed at all. In FIB, indexes of around 60% of the nodes are spread across 50-80% of the network. Clearly, in FIB indexes of all nodes are spread to good extent giving better accessibility to indexes. This is depicted in 3.36.

Looking closely at the dissemination of indexes would give us effectiveness of the approach. When a query reaches a node, node copies the indexes from the query to the local node. The result of this operation can be either new addition or update or just a duplicate index received. In figure 3.37, the % of those results is listed. The maximum of duplicates is found in flooding methods. In schemes with walkers as the carriers, fuzzy walkers scheme and depth-wise scheme have maximum duplicates. This gives us the idea that how the traffic is being utilised for disseminating the indexes. Surely flooding has more cases where the query reaches the same node more than once. In fuzzy walkers case, the walkers are directed towards a neighbour which have the maximum fuzzy probability value for finding the content it is looking for. The probability of selecting a set of neighbours more often than others is high. Therefore the same indexes may reach more than once. The best case here is floating indexes breadth-wise with random walkers. Since the neighbours are chosen randomly, there is high probability that each neighbour is equally likely to forward the query. In figure 3.38, the percentage of foreign indexes accumulated at each node is depicted. Although FI fuzzy scheme and FI depth-wise scheme have nearly the same percentage of traffic utilized but the percentage of indexes accumulated is much less compared to FI depth-wise scheme. This can be explained as: in FI fuzzy scheme, the walkers are guided towards the content. The average pathlength is around 1.5. That means the queries are answered very quickly. So the walkers end up at early stages. Therefore the spread of indexes is not as much.

Although FID scheme has uniform spread, spreading larger number of indexes yet its search performance is much below the FIB. Looking at tables 3.13, and 3.12, it can be noticed that FID has fewer successful queries and larger hop lengths. This indicates that in FID, the indexes accumulated at a node are predominantly that of few nodes where as in FIB they are mix of large number of nodes across the network. This is depicted in figure 3.36.

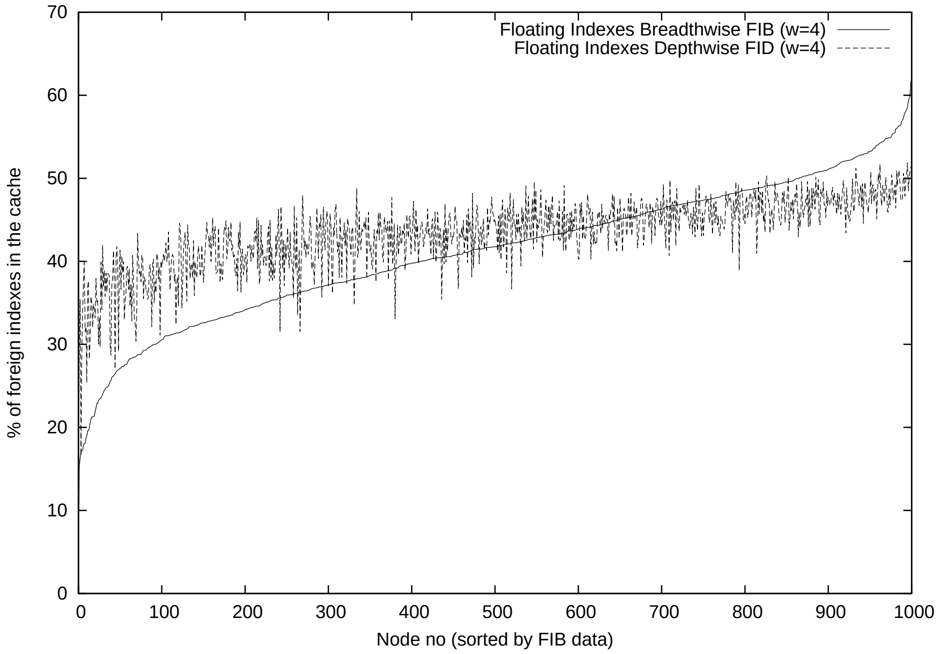
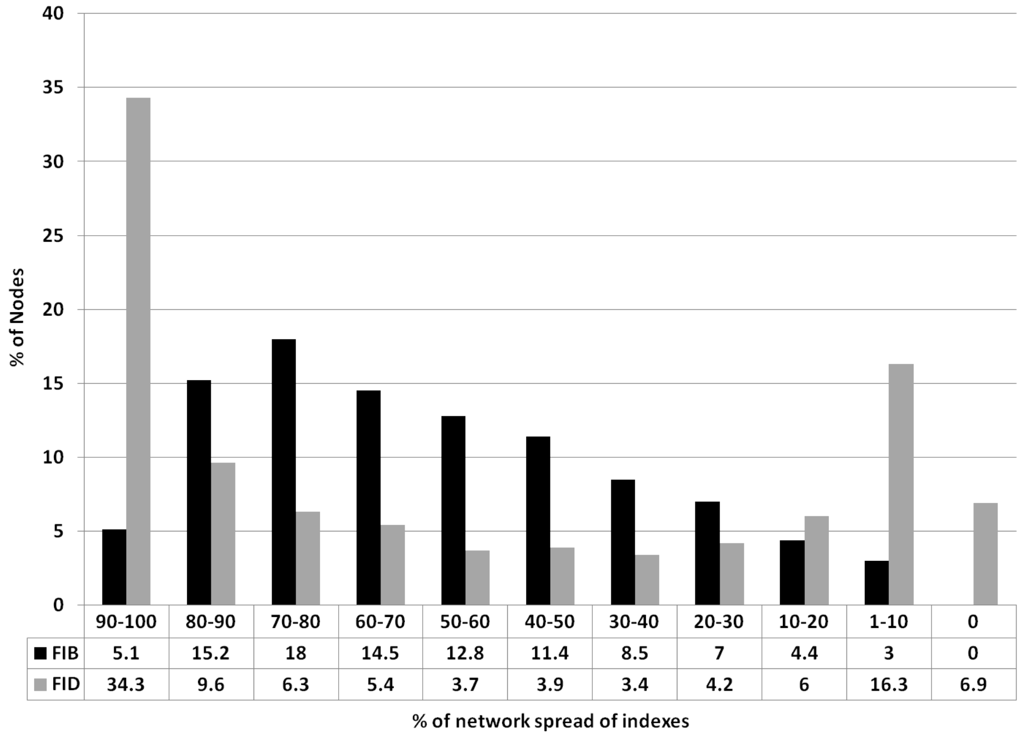


Figure 3.35: Spread of foreign indexes across nodes - FIB has standard deviation of 8.21% and FID has standard deviation of 4.28%

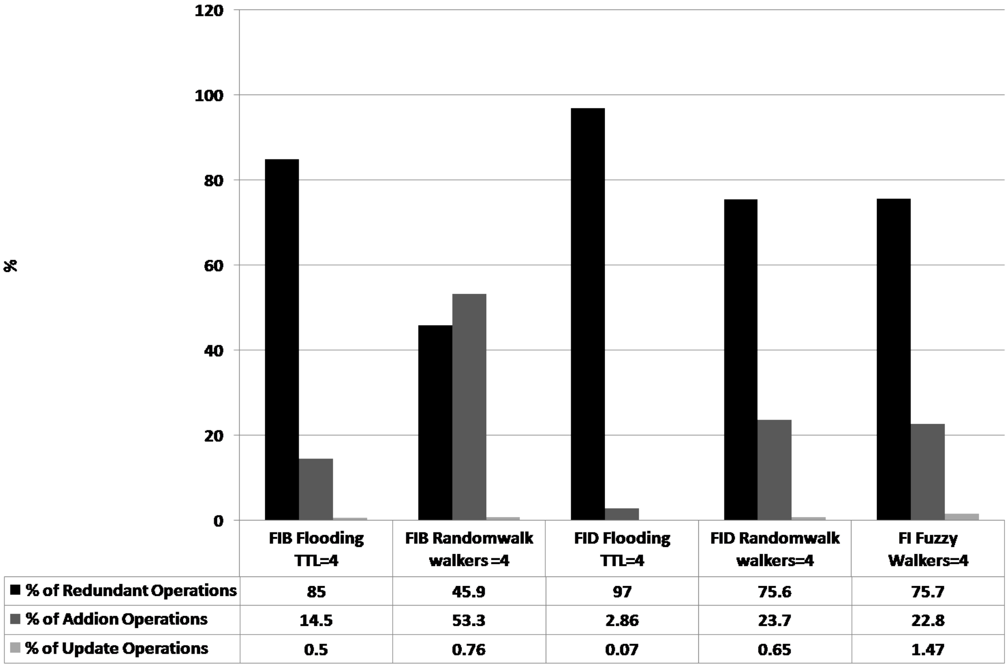


	90-100	80-90	70-80	60-70	50-60	40-50	30-40	20-30	10-20	1-10	0
■ FIB	5.1	15.2	18	14.5	12.8	11.4	8.5	7	4.4	3	0
■ FID	34.3	9.6	6.3	5.4	3.7	3.9	3.4	4.2	6	16.3	6.9

Figure 3.36: Percentage of network spread of indexes - x-axis: % of network spread, y-axis: % of nodes whose indexes are spread

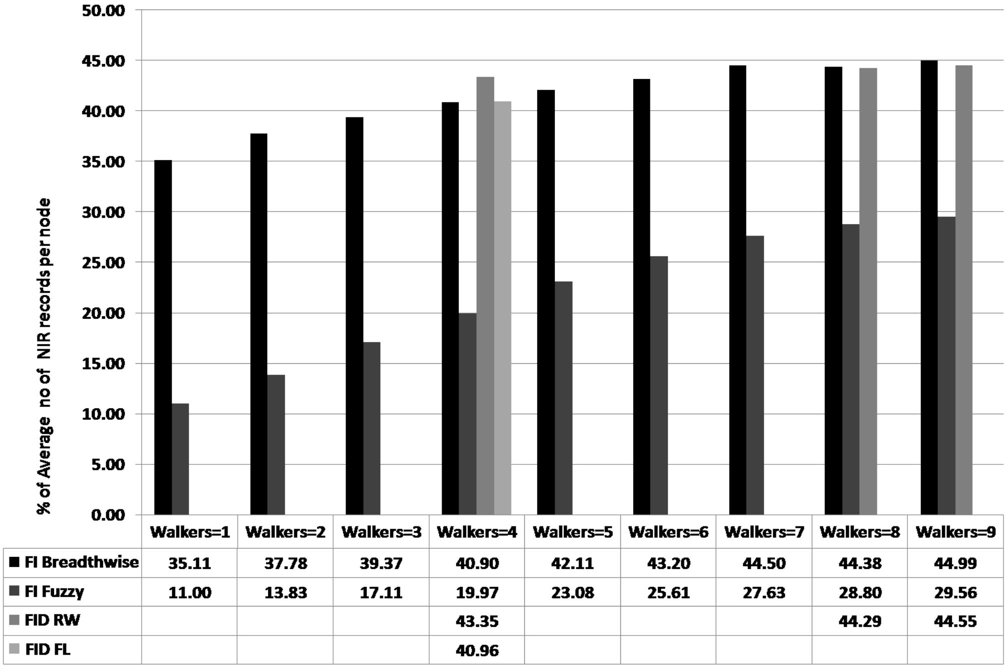
3.3.4.11 Adapting to Churn

To observe the churn effect, after 6000^{th} query onwards, 30 nodes are removed and 30 nodes are added at every 500^{th} query. The effect would be that objects, indexes stored at that node will be lost. When new nodes join, they would share new objects. So when nodes query for those items which are freshly added to the network, there will be surge of messages as that item would not have been indexed by now. Figure 3.39 depicts this behaviour in the network. At 6000^{th} query, there is sudden rise in the messages. But soon that settles down. The degree of such rises also decrease with the progress of queries. Figure 3.40 depicts the effect on pathlegths. At 6000^{th} query the gap between the two curves increases but soon after this gap becomes even. This indicates that proposed approach is quickly adapting to the churn scenarios.



	FIB Flooding TTL=4	FIB Randomwalk walkers =4	FID Flooding TTL=4	FID Randomwalk walkers=4	FI Fuzzy Walkers=4
■ % of Redundant Operations	85	45.9	97	75.6	75.7
■ % of Addion Operations	14.5	53.3	2.86	23.7	22.8
■ % of Update Operations	0.5	0.76	0.07	0.65	1.47

Figure 3.37: Percentage of cache operations for various schemes - each operation results in either addition, updation or duplicate detection



	Walkers=1	Walkers=2	Walkers=3	Walkers=4	Walkers=5	Walkers=6	Walkers=7	Walkers=8	Walkers=9
■ FI Breadthwise	35.11	37.78	39.37	40.90	42.11	43.20	44.50	44.38	44.99
■ FI Fuzzy	11.00	13.83	17.11	19.97	23.08	25.61	27.63	28.80	29.56
■ FID RW				43.35				44.29	44.55
■ FID FL				40.96					

Figure 3.38: Percentage of average foreign indexes at the local node - NIR refers to Node Index Records

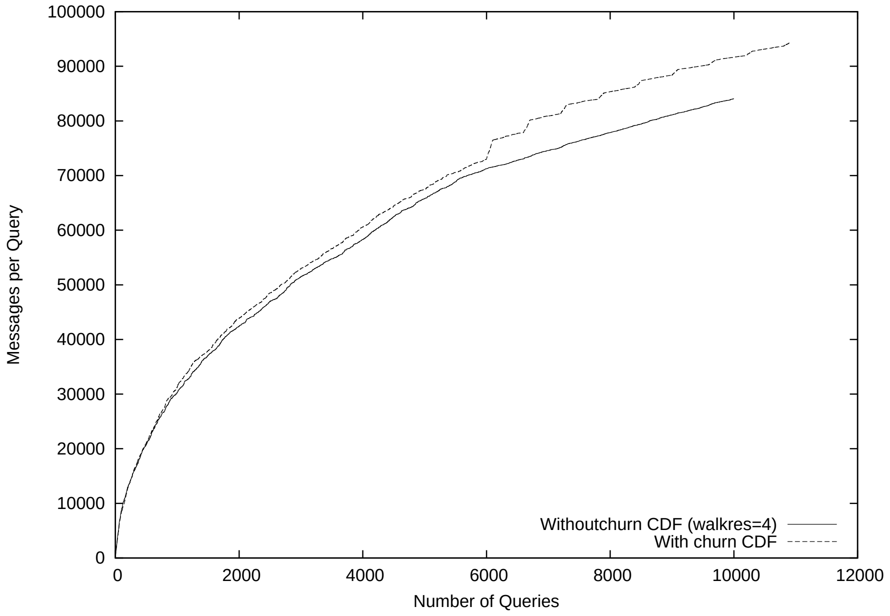


Figure 3.39: Effect of churn on Floating Indexes: message generation - churn is the scenario in which considerable number of nodes leave the system and new nodes join the system

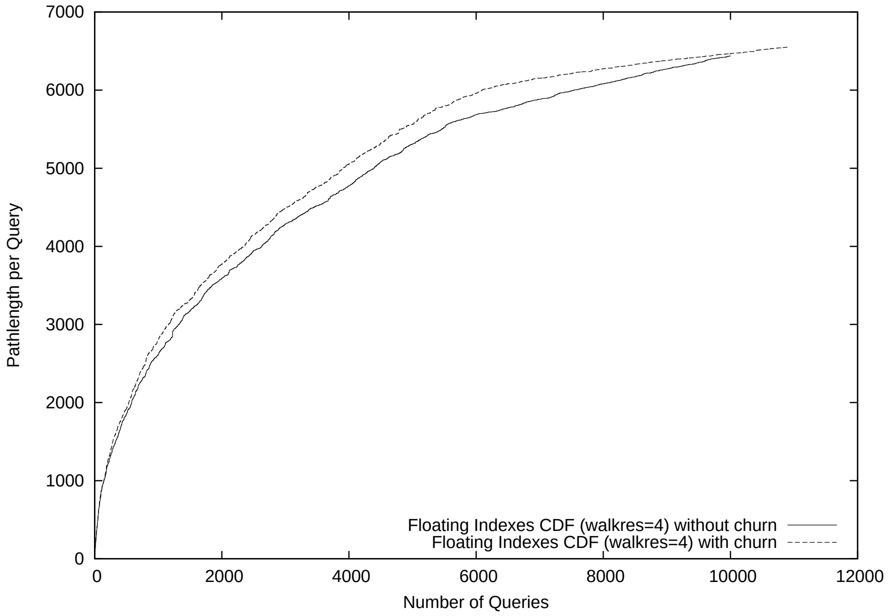


Figure 3.40: Effect of churn on Floating Indexes: pathlengths - churn is the scenario in which considerable number of nodes leave the system and new nodes join the system

3.3.5 Conclusion

The approach proposes an indexing scheme known as 'Floating Indexes' without using any extra messages. Various schemes within this approach are in detail discussed. In conclusion, Floating Indexes with random walkers has the best returns as far as index dissemination is considered. In comparison to attenuated BloomFilters method, there is an improvement of 53% in pathlengths, 97% reduction in message traffic, keeping the same success rate. Breadth-wise spreading of indexes through random-walk utilized 54% of the query traffic effectively to spread the indexes. This method has spread indexes of 77% of the nodes onto more than 50% of the network. This method has limitation that entire index propagation is dependent on query traffic.

3.4 Conclusion

In this chapter, two approaches are proposed. One is Fuzzy-probability based neighbour subset selection and the second is 'Floating Indexes' approach. Both have better gains over their counter parts. It was shown that fuzzy-probability approach has better forwarding policy over cosine-similarity and other methods. The improvement is in capturing the heuristics through classifying and fuzzifying the parameters. The major contribution of 'Floating Indexes' approach is that it doesn't need extra messages and has given very good results in utilizing query traffic for index dissemination.

It is also discussed about the effects of combining these two approaches as Floating indexes breadth-wise with fuzzy walkers. Although it has good returns, it is not effective in disseminating indexes which is the primary purpose of the second approach. We give the following conclusion on how to decide which approach to choose in peer-to-peer file sharing networks.

- **Context of application for Fuzzy-based neighbour subset selection approach**: The objects in the network fall into fixed set of categories and each object can be relatively rated on its popularity by the user. The number of objects in the network are not too high.

- **Context of application for Floating Indexes approach**: Network has extensive query traffic.

Since the contexts are not complimentary, these two approaches can be combined for better efficiency and responses at the cost of reduced index dissemination.

Chapter 4

Quality of Service (QoS) in Content Search

Search in peer-to-peer networks is measured by metrics generally grouped into efficiency metrics and quality-of-service (QoS) metrics. QoS metrics measure the search performance from the user's point of view. In this chapter, a search mechanism is presented that addresses QoS in search considering user preferences in terms of his node's link capacity, link latency, type of file etc.

4.1 Quality of Service (QoS)

4.1.1 QoS in Internet

Packet-switched networks like the Internet offer best-effort delivery of network traffic. This may be suitable for applications like FTP etc where traffic is elastic. The elastic traffic can tolerate delays, losses and adapt to congestion. But applications like video streaming etc can tolerate the latency only within a particular range. Such traffic is inelastic. There is rapid increase in usage of such applications in the Internet. Categorization of internet applications is shown in figure 4.1. QoS provides network services that are differentiated by their bandwidth, latency, jitter, and error rates. Collective effect of service performances that determine the degree of satisfaction by a user of the service.

IETF task forces have outlined schemes Integrated Services (IntServ) and Differentiated services (DiffServ) and also a Resource Reservation Protocol (RSVP). IntServ offers per-flow QoS which requires routers to maintain per-flow state information where as DiffServ offers per-class QoS which doesn't require routers to maintain state. Thus DiffServ is suitable for large-scale networks. RSVP acts as the protocol to signal resource reservation. This protocol is used by both IntServ and DiffServ schemes [Ferguson & Huston 1998]. IntServ provides guaranteed service and controlled-load service. Guaranteed service targets at hard real-time applications which need absolute provisioning. User specifies traffic characteristics (TSpec) and a service requirement. Controlled-load targets at applications which can adapt to network conditions within a certain performance window. DiffServ labels the packets with a priority and defines behaviours for routers to deal with packets different priorities.

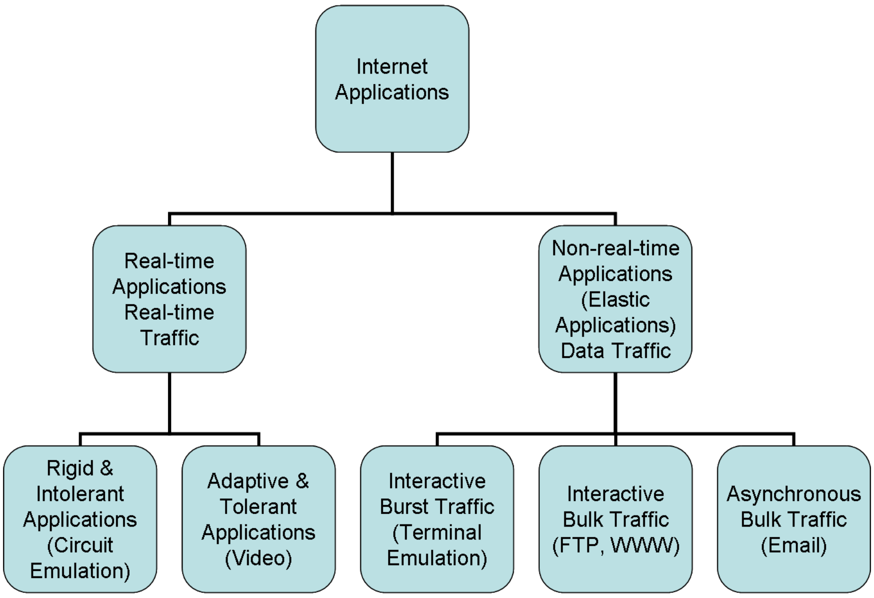

Internet Applications
Real-time Applications Real-time Traffic
Non-real-time Applications (Elastic Applications) Data Traffic
Rigid & Intolerant Applications (Circuit Emulation)
Adaptive & Tolerant Applications (Video)
Interactive Burst Traffic (Terminal Emulation)
Interactive Bulk Traffic (FTP, WWW)
Asynchronous Bulk Traffic (Email)

Figure 4.1: Internet applications classified by traffic - Source: [Kosiur 1998]

4.1.2 QoS in Peer-to-Peer Search

As discussed in Chapter 1, peer-to-peer networks are overlays built on Internet having topology and routing different from the underlying network. Forwarding a packet from peer B to peer A actually involves many links in the underlying network as shown in figure 4.2. So providing QoS in the underlying network doesn't automatically provide QoS in peer-to-peer overlay. QoS in the Internet is provisioned at the routers. But in peer-to-peer overlay the routing is carried out by the overlay nodes themselves with the help of underlying network. So provisioning QoS in peer-to-peer network involves the overlay nodes or nodes at the Internet edge.

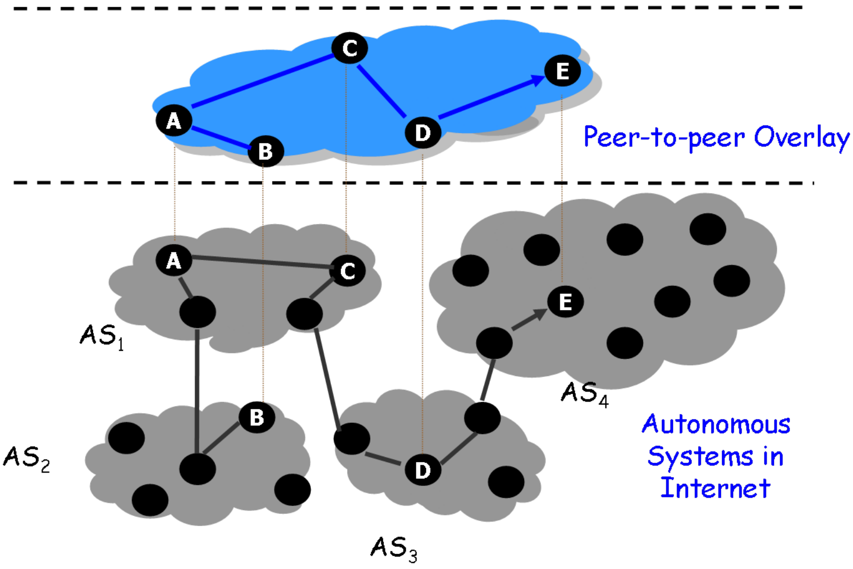


Figure 4.2: Forwarding in peer-to-peer overlay - Forwarding on one overlay link involves many links in the internet

[Ge & Cai 2008] have proposed a DiffServ QoS model for peer-to-peer file sharing systems. They have considered two classes of services namely standard service and premium service for standard users and premium users respectively. They have considered response time, throughput and warm-up time as QoS parameters. The idea is that the master node in the cluster labels each message with the priority and the nodes in the overlay forward the premium messages with high priority. Premium messages not only include the messages from premium users but also the system maintenance messages. This reduces the warm-up time. In this model, the differentiation is made between the users. But each individual user's requirements can be many-fold not just premium or standard. This is not addressed. [Zhang *et al.* 2009] proposed Web Ontology Language (OWL) for describing QoS classes in semantic service discovery. The QoS parameters used are connectivity, reliability and trust degree. These research contributions adapt the QoS schemes at the Internet level to overlay layer. An important consideration that has to be made here is that the characteristics of routers in the Internet and the characteristics of nodes in the peer-to-peer overlay are significantly different. Routers in the internet are special-purpose devices that are highly available, resourceful and tuned to give high performance routing throughput. The links connecting routers have huge bandwidths, low latencies and are highly available. The systems in peer-to-peer content sharing overlays are transient and have diverse network capacities. In section 4.1.2.1, the measured network characteristics of peers are discussed. This brings us to the point that while considering QoS in peer-to-peer overlays, distinction between whom to forward and whom not to forward is to be made. In this work, we consider this aspect.

QoS in search is defined by [Daswani *et al.* 2002] as provisioning user-perceived qualities such as number of results, response time, relevance, user satisfaction etc in their acceptable performance. The exact list of qualities or metrics and their acceptable range is application specific. These qualities or metrics are affected by network parameters such as bandwidth, latency, neighbour's reliability, search method used etc.

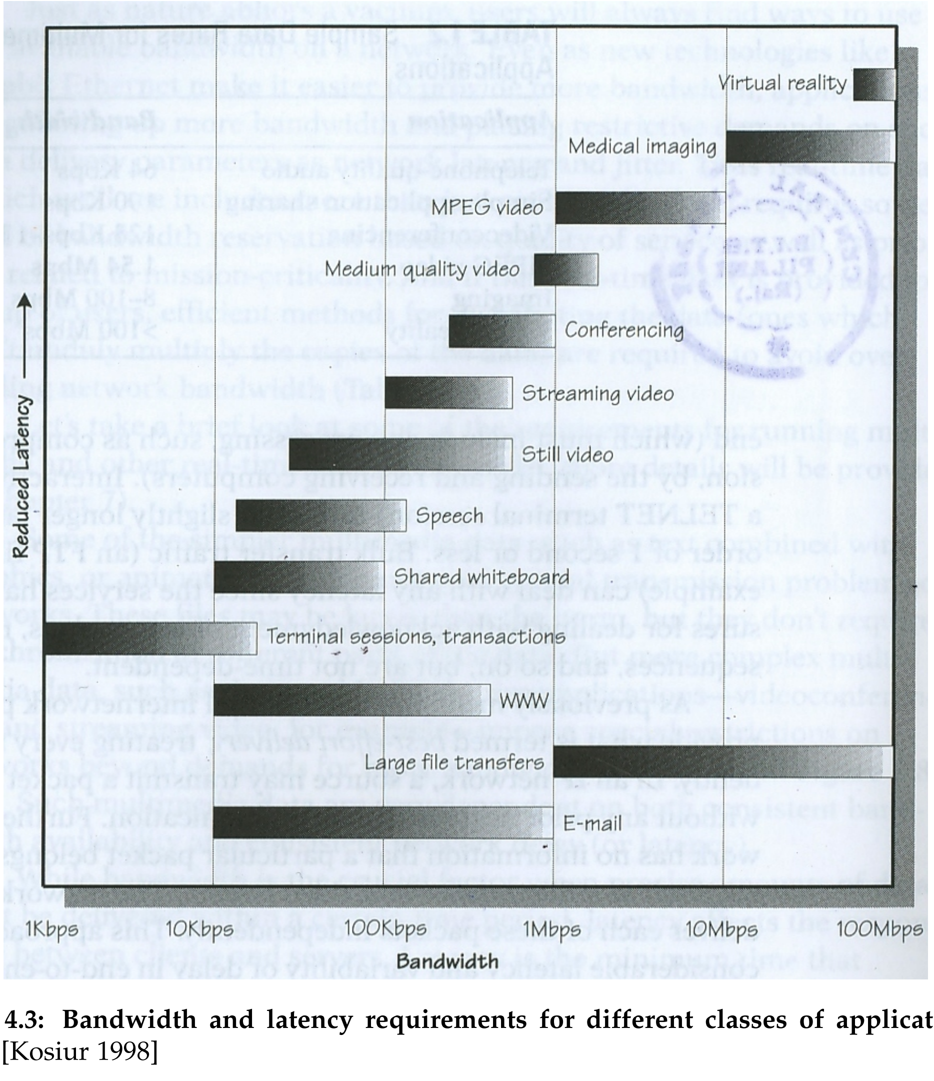


Figure 4.3: Bandwidth and latency requirements for different classes of applications - Source:[Kosiur 1998]

As shown in figure 4.3, the network applications have diverse requirements of bandwidth and latency. The user-interacting applications such as terminal sessions, speech, streaming etc require low latencies. In those applications requiring file-transfers, the concern is more about bandwidth. Overall, there is a judicious combination of bandwidth and latency levels for each application. In a peer-to-peer content sharing overlay, the content is distributed by downloading as well as streaming. In download, the entire file is first transmitted and then used. In streaming, the file is used while being transmitted. Even in streaming there are different network requirements for streaming different types of video files. Each method has its own QoS specifications.

In a content-sharing overlay, the content-search precedes the transmitting of content. In the search process, the resource-peer is discovered. The QoS requirements of content transmission are to be validated during the search process. This is different from the Internet where there is no host discovery process before transmitting content. The host address is already known to the content seeker. Requester directly sends a download request. But in peer-to-peer networks, the content is discovered first and then transmitted to the requester. The content is transmitted by the resource-peer along the same path as the path taken by the query to reach the resource-peer. In some networks like Gnutella, the content is transmitted through a direct TCP connection between the requester and the resource-peer. But in a system like Freenet, the content is transmitted along the query path. In Gnutella, the user anonymity is not preserved. But Freenet provides user anonymity by hiding the requester identity. The work in this thesis, assumes a model like Freenet for transmitting content.

[Li 2008] argues that peer-to-peer networks should serve heterogeneous needs of the users. Currently there are systems for providing file sharing, there are different systems for streaming, and there are different systems for IP telephony. He argues that for peer-to-peer in order to make best utilized there is a need to integrate all the services into a single network. In this work, a QoS model is proposed which can integrate variety of services.

4.1.2.1 Network Characteristics of Peers

Due to their large size and wide spread use of peer-to-peer overlays, the characteristics of the nodes are not uniform. Some nodes have high network capacity, but some have low capacity. [Saroiu *et al.* 2002a] analyse the characteristics of the nodes in Napster and

Gnutella in 2002. In figure 4.3, the range of bandwidth capacities of nodes is presented. It can be observed that most of the nodes (65%) use broadband connections (cable modems, DSL, T1, T3) and the rest use dial-up modems. The home users mainly use modems, cable modems and DSL.

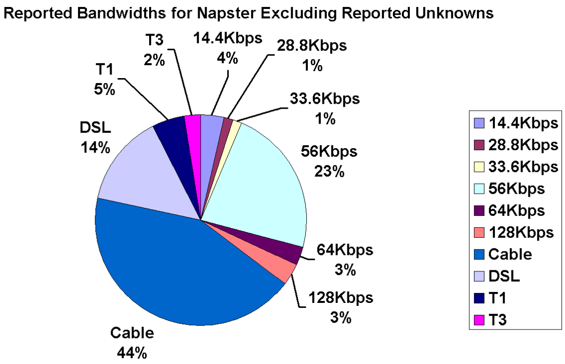


Figure 4.4: Heterogeneous nature of bandwidth capacities in Napster - Source: [Saroiu *et al.* 2002a]

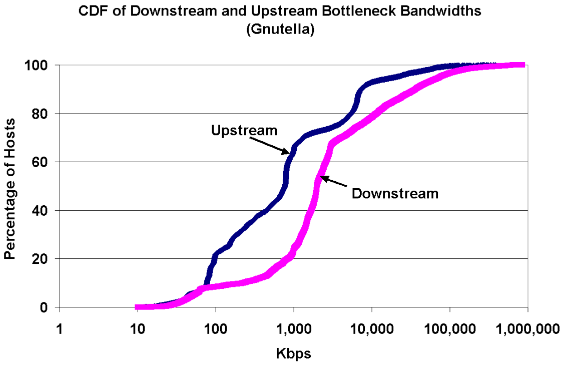


Figure 4.5: Bandwidth distributions of nodes in Gnutella - Source: [Saroiu *et al.* 2002a]

In figure 4.4, a pie chart of bandwidths used by user is presented. It can be observed in figure 4.5 that while 92% of the participating peers have downstream bottleneck bandwidths of at least 100Kbps, only 8% of the peers have upstream bottleneck bandwidths of at least 10Mbps. Moreover, 22% of the participating peers have upstream bottleneck bandwidths of 100Kbps or less. This variation poses a challenge to providing QoS on downloads. Similarly in figure 4.6, approximately 20% of the peers have latencies of at least 280ms, whereas another 20% have latencies of at most 70ms. For streaming, latency is an important consideration as the timing at which audio or video frames arrive decides whether they are worth for playing.

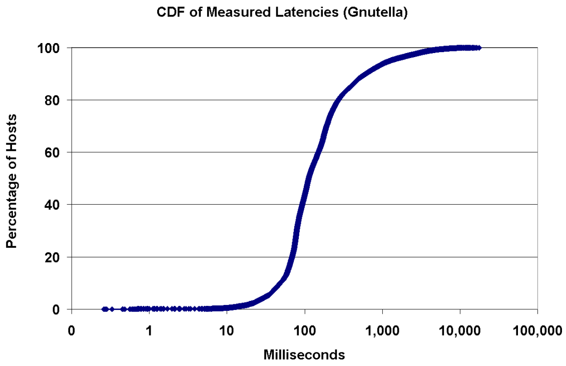


Figure 4.6: Latency distribution in Gnutella Source - Source: [Saroiu *et al.* 2002a]

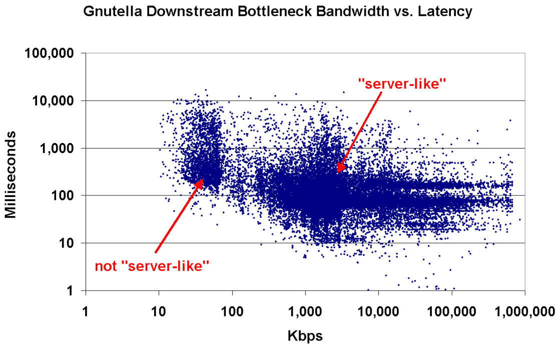


Figure 4.7: Latency and bandwidth distribution in Gnutella - Source: [Saroiu *et al.* 2002a]

In 4.7, it can be observed that the there are two clusters having low bandwidth and high latency systems (20-60Kbps, 100-1,000ms), and systems with high bandwidth and low latency (1,000Kbps, 60-300ms). They correspond to modems and broadband connections.

4.1.3 Summary

The discussion is summarized as follows.

- Forwarding of a packet on one link at overlay level may actually involve many links in the underlying network. Therefore QoS provisioning should give specific attention to nodes in the overlay.
- The nodes in the peer-to-peer overlay act as routers at the overlay level. These nodes have different characteristics from the routers in the underlying network.
- There are diverse requirements of bandwidth and latency across the network appli-

cations. The choice of the links according to these parameters has direct affect on the user perceived qualities (QoS) of the search mechanism.

- The peer-to-peer systems are built at different times for different purposes. But there is no integrated access for all such services in one single network.

4.2 Proposed Solution

The scope of the problem is defined as below.

- The overlay network is content sharing network where the content can be of large files like video, and music which can be both downloadable and/or streamed. Also content may mean any documents like pdf, word documents, power point slides etc.
- In such a network, to provide quality of service to the user, the network requirements for the type of file the user has requested must be differentiated and accordingly provided.
- The QoS as provided in DiffServ can't fully serve such requirements, the reasons being: the resource node is not known in advance and the nodes capabilities are too diverse. There is difference between routers in IP layer and the overlay nodes in terms of their capability and the availability. Therefore taking any path to reach the resource node may include nodes whose capacity and availability is too much below the requirements. Although the packets may be forwarded with high priority, but the overall throughput may not meet the requirements.
- Therefore the problem we are addressing here is to provide QoS requirements based on the file type in the search by path selection.

4.2.1 QoS Parameters

[Kwok & Yang 2004] have done a study on Gnutella network identifying the characteristics of the users of the network. Table 4.1 shows the percentages of different kinds of files that are requested by the users. Table 4.2 summarizes the results into well known file types.

Table 4.1: File extensions in Gnutella Source:[Kwok & Yang 2004]

File Extension	Percentage (%)
mp3	22.76
Avi	17
Mpg	9.72
Zip	1.64
Mpeg	1.61
Jpg	1.08
Asf	0.81
Divx	0.69
Ra	0.64
Rm	0.42
Pdf	0.39
Exe	0.34
Rarm	0.34
Ps	0.17
Mov	0.17

Table 4.2: File types searched for in Gnutella - Source: [Kwok & Yang 2004]

File Category	Percentage (%)
Video	30.01
Audio	24.15
Compressed file	2.01
Graphic	1.12
Document	0.67
Software	0.36
Other	41.69

It can be observed that video and audio are the most requested type. This gives us an idea that in peer-to-peer content sharing networks users have varied interests and thus expect varied services from the network.

The services can be classified into 3 categories.

1. Downloading large files such as music, video, software etc

2. On-demand viewing of videos and listening to music etc

3. Retrieving documents like pdf, doc, zip etc and graphics such as jpg, gif etc whose size is small

These services have different network requirements. The requirements vary for each type of service and are to be specified in terms of the overlay network parameters namely overlay link bandwidth, overlay link latency and object location probability. Table 4.3 shows the mapping between the user expectations and the corresponding QoS conditions to be fulfilled. The QoS metrics applicable in a file sharing system are bandwidth, latency and object location probability. Bandwidth (B) is non-additive parameter in the end-to-end bandwidth computation. It is taken as the minimum of the available bandwidths of the links comprising the path. Latency (L) is an additive parameter for computing end-to-end latency. It is the sum of individual link latencies of the path. Object location probability (F) is the probability of finding the object in a particular neighbour or its further neighbours.

Table 4.3: Mapping service requirements to QoS parameters (B:Bandwidth, L:Latency, F:Object Location Probability)

Services	**Qualities Expected by User**	**QoS Parameters**		**Overlay Path QoS (Parameter Priority)**
		Overlay Link Qualities	**Overlay Node Qualities**	
Downloading large files	Fast file download	High Bandwidth		$B > F \geq L$
Playing a on-demand/live video/audio	Uninterrupted Play	Low Latency		$L > F \geq B$
Finding and retrieving documents, images etc	Quick search and retrieval		Object Locality	$F > B = L$

The relative importance of parameters is chosen by the user for particular search transaction. They may be available as presets [1] in the user application.

4.2.2 QoS Path Selection in the Overlay

Providing QoS parameters for different services requires selecting overlay path that conforms to the QoS condition. In IntServ QoS provisioning, a per-flow state is maintained in the routers and reservation of resources is made [Ferguson & Huston 1998]. But this method is not suitable for large overlays because it is not scalable, and there is huge variation among the capacities of the overlay nodes as described in 4.1.1. Every overlay node can't be expected to provide the same facility. So in our method, suitable overlay nodes are chosen to form a path up to the resource-node. There is no QoS-state information maintained in the overlay nodes. Choosing a right node to form the path involves choos-

[1] Preset is a set of parameter values set and fixed by the user which will be default for the rest of the requests

ing one node at each hop among many neighbour nodes such that the chosen link satisfies multiple criteria QoS metrics like required bandwidth, latency, delay, packet loss etc.

This problem is commonly known as Multi-Constrained Path (MCP) selection problem which is NP-complete [Wang & Crowcroft 1996]. There are several algorithms in literature for finding feasible paths satisfying multiple constraints. [Chen & Nahrstedt 1998] showed that the problem can be modified by scaling down one weight of each link to a bounded integer x and that solving this problem by Dijkstra's or Bellman-Ford algorithm will give the solution to the original problem. To find a feasible path, x should be very large leading to huge time complexity $O(x^2N^2)$ in case of Dijkstra's algorithm is used. [Yuan 2002] used two heuristics, the limited-granularity heuristic and limited-path heuristic and proposed Limited Path Heuristic Algorithm (LPHA) that has time complexity as $O(x^2NE)$ where x is number of non-dominated paths to be maintained at each node. TAMCRA(SAMCRA) proposed by [Neve & Mieghem 2000] uses a non-linear path function and computes feasible paths by an extended Dijkstra's algorithm.

[Jaffe 1984] proposed a linear link-cost function that combines link costs in proportion to their weights. This single aggregate metric is used to find shortest path using any shortest path algorithm. Here the algorithm assumes the availability of global state at the current node. [Mieghem *et al.* 2001] proposed hop-by-hop QoS (HbHDBO) routing algorithm assuming that global state is available at each node. They use non-linear weight function for the path. This algorithm makes use of TAMCRA proposed by [Neve & Mieghem 2000]. [Sobrinho 2003] uses an algebraic approach to investigate the path optimization problem. He provides properties of path weights that need to be satisfied to avoid loops and attain optimal paths.

All the above algorithms assume that there exist a global topology and link-state information at the current node which is not a scalable approach for large-scale peer-to-peer networks. [Li & Garcia-Luna-Aceves 2006] proposed a distributed approach for finding feasible paths satisfying multiple constraints. In their work every node stores a metric to every destination through every neighbour. The space complexity at each node is $O(xkN)$ where x is the number of routes to keep, k is the number of neighbours, and N is the number of nodes in the system. As the network grows large, state per node also linearly grows. This is not desirable in the large peer-to-peer overlays.

All the above algorithms know the destination in advance. Their aim is to find a feasible path given a source and destination. But in peer-to-peer file sharing systems the destination is not known in advance and it has to be searched out. There can be multiple destinations depending on the number of replicas of the target object. Searching a destination is coupled with the availability of a path that satisfies the constraints from source to destination. Therefore the search procedure is to be augmented with the function of finding the path that satisfies the given constraints. The distributed search procedure needs to locate destination in a way that it finds a path that satisfies the given constraints.

In the above algorithms there are two ways of path selection.

- One way is to select paths satisfying the given constraints. For example: Suppose there are two constraints: Latency $L \leq \alpha$ and bandwidth$B \geq \beta$. Now every link on a path is tested to see that these two constraints are satisfied. If there is a path from source to destination satisfying these constraints at every link then that path is said to be feasible path. Among such feasible paths selection of a path that is optimized with respect to a metric (usually hop count) is known as Multi-constrained optimal path (MCOP) problem.

- Another way is to combine the link metrics into a single aggregated-metric and choose the shortest path with respect to aggregated-metric. The individual metrics can be combined in linear and in non-linear ways. For example: B_j and L_j are bandwidth and latency measured at node n to neighbour j. w_B and w_L are the weights assigned to bandwidth and latency respectively. Then aggregated metric is computed as

  $$m_j = w_B \times B_j + w_L \times L_j$$

  This aggregated-metric is known as the link-weight or link cost. It is used in the computation of shortest-path. Path computed using aggregated metric may not have all the links satisfying the given constraints. Most of the proposed algorithms use aggregated metric to compute the paths. Out of these paths, some may be feasible paths.

Even with global state (topology and link-state) of the network available at one place, to compute a feasible path is known to be NP-complete problem. That's why, the above mentioned algorithms use heuristics to find optimal paths. To compute a feasible path in a distributed way is more difficult due to unavailability of network global state. It creates more uncertainty in path selection. Therefore our proposed solution has the following characteristics.

- Path selection is performed while searching for the object
- Path selection is done using aggregated cost in a hop-by-hop manner
- It doesn't require global state at the node
- It doesn't guarantee feasible paths but finds candidate paths according to the priorities the user has set. There is provision to choose optimal candidate paths according to an end-to-end aggregate-metric. It is different from DiffServ in that it is aiming at path selection instead of class-based prioritized forwarding. Class-based prioritized forwarding or guaranteed QoS as in IntServ can be utilized once a candidate path is identified from source to destination. The problem being addressed here is to identify optimal candidate paths according to the type of search query.

Basically, the path selection based on aggregate-metric is a greedy approach. Every time the link with minimum aggregate-metric is selected. If the aggregate is based on local neighbourhood, it doesn't guarantee reaching the global optimal path. If the aggregate has some basis of global end-to-end characteristic then there is a scope for achieving the global optimum if that metric satisfies greedy-choice property and optimal-substructure property[Cormen *et al.* 2009]. In our solution, the aggregate is computed on local bandwidth, object location probability and local latency. The probability of locating the object is derived from pre-computed attenuated bloom filters. This is not entirely a local metric but has limited global scope to the extent of levels in attenuated bloom filter. The aggregated-metric doesn't satisfy greedy-choice property and thus it can't guarantee global optimum path selection. But the quality of the path selection or how closely it fulfils the user expectations is dependent on the cost function for calculating the aggregate-metric. The solution is tested with one linear cost function and two non-linear cost functions.

The problem of choosing the best alternative that satisfies multiple criteria among many alternatives in literature is known as Multi-Criterion Decision Making (MCDM). There are methods based on distance, outranking, and utility [Raju & Kumar 2010] for solving MCDM problems. We used Weighted average, Compromise Programming, and Technique by Order Preference by Similarity to an Ideal Solution (TOPSIS) as the cost functions to compute aggregate-metric.

4.2.3 Object Search cum QoS Path-Selection Algorithms

This section describes the search procedure in general and procedures required for path selection. First we describe the system model followed by the actors of the system and their roles are explained.

4.2.3.1 System Model

The model is a fully decentralized peer-to-peer unstructured network. There are N nodes numbered 0 to N-1 in the overlay as shown in the figure 4.8. Each node n has a set of neighbors G. Each node n regularly probes and stores bandwidth B and latency L of the neighbour links. B_j refers to the bottleneck-bandwidth measured at n to neighbor j. L_j refers to latency measured at n to neighbor j. $B_{\max}$ refers to the theoretical maximum available bandwidth in the network. $L_{\max}$ refers to the theoretical maximum latency in the network. $B_j^{normalized}$ refers to the normalized bandwidth and $L_j^{normalized}$ refers to the normalized latency of the link.

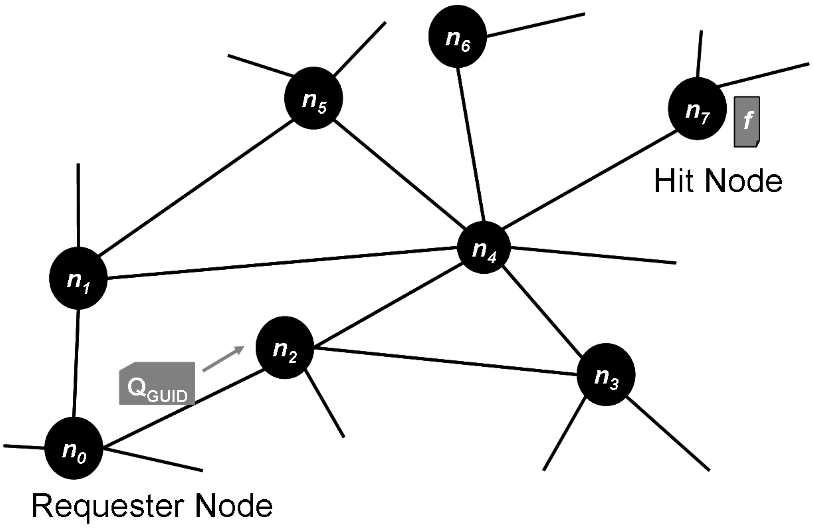


Figure 4.8: System model for QoS-constrained search - n_0 is searching for f available at n_7

Files of various kinds like music, video, documents etc are shared by the individual nodes. Same file may be available at multiple nodes. Depending on the popularity of the file, the file soon spreads to other nodes. [Chu *et al.* 2002] observed that this distribution follows Zipf distribution. Requester node n sends a query Q_{guid} searching for file f. Each query is associated with a globally unique identifier *guid*. The user specifies the QoS requirements. While forwarding the query the next hop destination is decided by these QoS requirements. At each node, a guid-cache is maintained which contains the mapping between *guid* and the node which has sent the query Q_{guid}. Any node if it finds a matching object with it, it will form a QueryHit reply QH_{guid}. QueryHit travels in the same path taken by the query Q_{guid}.

The following are the participants or executors of the different procedures in a path-selection process.

- Requester Node
- Forwarding Node
- Intermediate Node
- Query Hit Node

4.2.3.2 Requester Node

The search is triggered by the user. This node initiates the query. The query carries the following items:

- *guid*
- keywords
- hop count
- weights for bandwidth, latency and object location probability
- path weight or cost

The weights for the QoS parameters are computed based on the priorities chosen or set by the user. The weight computation is performed by Analytical Hierarchical Process (AHP) method [Saaty 1994]. The details of it are explained with examples in the coming section. The path-weight parameter in the query accumulates the link weights. This parameter is returned by QueryHit node through QueryHit message to Requester node. Requester Node chooses the best path as the path with minimum path-cost. The requester node executes procedure *searchFile()* upon receiving its user search request. This procedure is presented in figure 4.9.

```
searchFile(Preset: $P$, KeyWords: $KW$)
begin
    Query: $Q$;
    Timer: $t$;
    Q.guid ← generateGuid();
    Q.hopCount ← MAX_HOPS;
    Q.keywords ← $KW$;
    Q.weights ← getWieghts($P$);
    Q.pathCost ← 0;
    Q.path ← $\Phi$;
    start timer $t$;
    processQuery($Q$, currentNode);
end
```

Figure 4.9: Requester node initializing query upon users search request

Requester node waits for QueryHit messages until the timer expires. This is depicted in procedure *processQueryHit()*. Once the timer expires, it sorts the QueryHit messages based on their path cost and the minimum-cost path is chosen for sending content request. This procedure is presented in figure 4.10.

4.2.3.3 Forwarding Node

Forwarding node refers to any node in the network that receives the query, processes and forwards it. Every node has a data structure that stores the bandwidth, latency and object location probabilities of the overlay links connecting to neighbours. The node probes the neighbours periodically to keep these values uptodate. The method for computing bandwidth and latency is described in later sections. Upon receiving a query, the node executes the procedure *processQueuedQueries()* shown in figure 4.11. For each query in the sorted queue, this procedure calls *processQuery()*. The *processQuerey()* is presented in figure 4.12.

```
processQueryHit(QueryHit: QH)
begin
    Node: dest;
    ResultSet: R;
    guidCache: C;
    Timer: t;
    /* guidCache C has mapping between guid received and node
    which sent it                                            */
    dest $\leftarrow$ get sender address from C looking up QH.guid;
    if dest = thisNode then
        R $\leftarrow$ R $\cup$ { QH.queryHitNode, QH.pathcost, QH.path};
    else
        send QH to dest;
    end
    if isExpired(t) then
        R $\leftarrow$ sortResultsByPathCostAsc(R);
        /* the first entry in R, i.e, R[0] has the least-cost
        path                                                 */
        Send Content Request along the path specified in R[0].path;
    end
end
```

Figure 4.10: Procedure for processing QueryHit messages

```
processQueuedQueries(Queue: U);
begin
    /* sort the queries in ascending order based on the cost
    accumulated so far                                       */
    U $\leftarrow$ sortbyCostAsc(U);
    while !isEmpty(U) do
        /* the first entry in U, i.e, U[0] has the least
        accumulated cost                                     */
        processQuery(U[0].query, U[0].sender);
        remove $0^{th}$ entry from U;
    end
end
```

Figure 4.11: Procedure for processing queued Query messages

```
processQuery(Query: Q, Node: sender);
begin
    GuidCache: C;
    Result: R=$\phi$;
    Files: F;
    QueryHit: QH;
    Query: CQ;
    NeighbourSet: G;
    CostSet: $T = \phi$;
    C ← loadGUIDCache();
    if Q.guid found in C then
        return
    end
    foreach file f in F do
        if match(f.keywords, Q.keywords) = true then
            R ← R ∪ f
        end
    end
    if R != Φ then
        QH.guid ← Q.guid;
        QH.R ← R;
        QH.pathCost ← Q.pathCost;
        QH.path ← concat(QH.path, address of thisNode));
        send QH to sender;
    else if Q.hopCount = 0 then
        return
    else
        C ← C ∪ {Q.guid, sender};
        foreach neighbour g in G do
            if g != sender then
                /* computeCost() computes the cost of a link
                associated with neighbour g.  It uses cost function
                discussed later                                    */
                T ← T ∪ {g, computeCost(Q, g)};
            end
        end
        /* sort the queries in ascending order based on costs of
        neighbour links                                            */
        T ← sortbyCostAsc(T);
        for i:1 to $count(G) \times \frac{1}{Q.popularity}$ do
            if T[i].g != sender then
                CQ ← clone(Q);
                Q.hopCount ← CQ.hopCount - 1;
                Q.pathCost ← Q.pathCost + T[i].cost;
                CQ.path← concat(CQ.path,address of thisNode);
                send CQ to T[i].g;
            end
        end
    end
end
```

Figure 4.12: Procedure for processing Query message

One of the features in the procedure given in 4.12 that makes it work is *guid* or globally unique id assigned to every message in the system. To avoid loops in forwarding the queries, node on the query path caches the *guid* of the query and sender of the query. Next time if a query arrives with the same *guid*, it is rejected. This way the system prevents looping in forwarding. This check may have a negative influence in QoS path selection. This can be illustrated with an example. Consider the diagram shown in figure 4.13. Node n_0 is initiating a query. We can observe that node n_4 is receiving the same query from n_1, n_2, n_3 and n_5. Whichever query reaches first at n_4, that will be considered and others will be discarded. This prevents looping. At node n_4, if query from n_2 comes first, then all other queries are discarded although their path-cost is lesser. In order to reduce this negative effect, it is proposed that before processing the queued queries, the queue is sorted by path-cost. This will ensure that the minimum path-cost query is considered first. In the procedure given in figure 4.11, it can be observed that the queued messages are sorted based on their path-cost.

Also it can be observed that the procedure computes cost for each neighbour except the one from which it has received the query. Then it sorts the cost-vector. The query is forwarded to the least cost links. The number of neighbours considered for forwarding is inversely proportional to the popularity of the file requested. If the file is popular, then there must be many replicas in the system. If it is unpopular, then it is forwarded to larger number of neighbours so that the coverage will be wider and may include the unpopular file.

4.2.3.4 Query Hit Node

Query Hit node is the node which has found one or more matches for the query. This node stops forwarding the query further. It makes the QueryHit message and copies the parameters from query message. The query hit message is sent over the same route through which the query has come. It assigns the same *guid* to QueryHit message as that of Query message. This is required for forwarding via the same path through which the query has reached the current path. This is shown in the algorithm given in 4.12 from lines $18 - 23$.

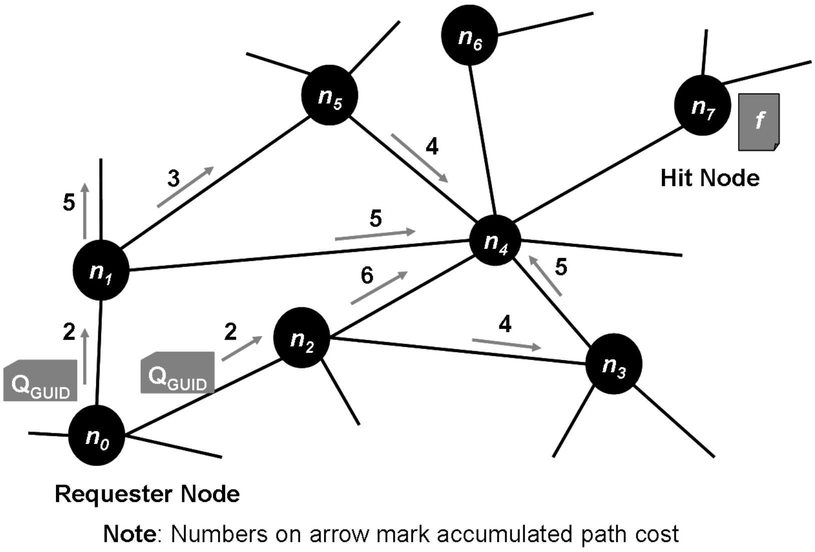


Figure 4.13: Effect of looping guard-condition on QoS path selection - guid is used to prevent looping in forwarding

4.2.3.5 Intermediate Node

Intermediate node refers to the node on the path of the QueryHit message. When this node receives the QueryHit message, it retrieves the destination node address from *guid*-cache. The QueryHit message is forwarded to destination node. This procedure is presented in figure 4.10.

4.2.3.6 Complexity Analysis

Time complexity: The time complexity to evaluate a single query at a node is as follows. F refers to the number of files to be searched at the local node and N refers to the number of neighbours. File search can be made constant or logarithmic time using indexing techniques. The costs are computed using different methods mentioned in section 4.12. The costs are sorted using a sorting technique. If we use linear search for files, and weighted-average method for cost computation and merge sort for sorting and flooding like forwarding then we have the expression for worst-case time complexity as given in equation 4.1.

$$O(F,N) = O(F) + O(1) + O(N \log N) + O(N) \tag{4.1}$$

$$O(F,N) = O(F) + O(N \log N) \tag{4.2}$$

If the cost computation method is TOPSIS or Compromise-programming, then

$$O(F,N) = O(F) + O(N) + O(N \log N) + O(N) \tag{4.3}$$

$$O(F,N) = O(F) + O(N \log N) \tag{4.4}$$

Space complexity: The state maintained at each node comprises

1. bandwidth, latency, probability vectors: for N neighbours, the space complexity is $O(N)$.
2. guid cache: the size of cache is computed as given below.

Assume that the guid-cache expiration time is δt. α is the query-rate per δt. We would like to calculate the extent of guids to be stored at the node in one cycle of expiration time. Assuming that every node has d as the node-degree and every query has hop-count limit as h and δt is maximum time for a query to travel h hops and return to the same node. Then during the interval δt the number of queries reaching to node are

$$\begin{aligned} Q_{\delta t} &= 2 * (\alpha_t + (\alpha * d)_{t+1} + (\alpha * d^2)_{t+2} + (\alpha * d^3)_{t+3} + ... + (\alpha * d^h)_{t+h}) \\ Q_{\delta t} &= 2 * (\alpha(1 + d + d^2 + d^3 + ... + d^h)) \\ Q_{\delta t} &= \left|\tfrac{2\alpha(1-d^{h+1})}{1-d}\right| \end{aligned} \tag{4.5}$$

$$O(\alpha, d, h)_{\delta t} = \left|\tfrac{\alpha(1-d^{h+1})}{1-d}\right| \tag{4.6}$$

The space complexity considered above is proportional to d^h. This is true in the case of flooding where every query is forwarded to all neighbours. But in the current solution the query is forwarded to only few neighbours, i.e., the number of neighbours is inversely proportional to the popularity p of the object being searched for. p is the index in 0-100 range. The lesser the value of p, more popular the object is. In that case the space complexity is proportional to $\left(\frac{p}{100} * d\right)^h$. We can notice that the space complexity does not involve the network size. A small reduction in d will result in huge gains.

If we take hop-count to be 7, α as 0.034, d as 15 and p is 30 then the size of *guid*-cache will be

$$= \frac{2.034(1 - 4.5^8)}{14} = 816.72$$

If we consider every guid occupying 8 bytes then the space requirement for *guid*-cache in the worst-case comes out to be $= 6.380kb$ where as in the case of flooding it would have been $= 97254kb$

4.2.4 Weight Calculation

The requester node calculates weights based on the user priorities or presets of priorities. The weight calculation uses Analytical Hierarchal Process (AHP) technique [Saaty 1994]. This method is helpful in assigning relative weights to each criterion based on subjective judgements. The input for this method is a pair-wise comparison matrix for each criterion. The method computes eigenvector corresponding to the maximum eigenvalue. The table 4.4 gives a nine-point scale of relative importance. User gives priorities as per the scale in this table.

For example, for downloading a large file, sample priorities are shown in Table 4.5. Bandwidth is given very strong importance over latency and strong importance over object location probability. Object location probability is given moderate importance over latency. The weights computed for this matrix using AHP are 0.73, 0.08, 0.19 assigned to bandwidth, latency and object location probability respectively.

Similarly for streaming search, a sample preset of priorities is shown in 4.6. The weights computed for bandwidth, latency and object location probability are 0.08, 0.73, and 0.19 respectively.

Similarly for quick-find search, sample preset of priorities is shown in 4.7. The weights computed are 0.1, 0.1 and 0.8 respectively.

Power method is used to compute eigenvalues. The maximum eigenvalue ($\lambda_{\max}$) is used to compute the eigenvector to determine the weights. The consistency index (CI) is defined as

$$CI = \frac{\lambda_{\max} - N}{N - 1} \tag{4.7}$$

Random Index (RI) is the consistency index of a randomly filled matrix of size N. The consistency ratio (CR) is defined as

$$CR = \frac{CI}{RI} \tag{4.8}$$

The consistency ratio value less than 0.1 is considered as acceptable [Saaty 1994]. The CR

Table 4.4: Saaty's nine-point scale for relative importance -[Raju & Kumar 2010]

Stage of Scale	Definition	Characteristics
1	Equal importance	Two criterion contribute equally
3	Moderate importance	Experience and judgement moderately favour one criteria over another
5	Essential or strong importance	Experience and judgement strongly favour one criteria over another
7	Very strong importance	A criteria is strongly favoured and its dominance demonstrated in practice
9	Extreme importance	The evidence favouring one criteria over another is of the highest possible order of affirmation
2, 4, 6, 8		When compromise is needed
Reciprocals		If criteria 1 has one of the above numbers assigned to it when compared with criteria 2, then criteria 2 has the reciprocal value of 1/2 when compared with criteria 1.

Table 4.5: A sample of relative importance to QoS-parameters for fast-download preset- fast-download preset is for downloading large files which requires huge bandwidth

	Bandwidth	**Latency**	**Object Location Probability**
Bandwidth	1	7 (Very Strong Importance)	5 (Strong Importance)
Latency	$^1/_7$	1	$^1/_3$
Object Location Probability	$^1/_5$	3 (Moderate Importance)	1

Table 4.6: A sample of relative importance to QoS-parameters for streaming multimedia preset-streaming preset is for viewing audio/video files which requires low latency

	Bandwidth	**Latency**	**Object Location Probability**
Bandwidth	1	$^1/_7$	$^1/_3$
Latency	7	1	5
Object Location Probability	3	$^1/_5$	1

Table 4.7: A sample of relative importance to QoS-parameters for quick-find search preset-quickfind preset is for retrieving small size files such as pdf,doc, or images where locating them is a priority

	Bandwidth	**Latency**	**Object Location Probability**
Bandwidth	1	1	$^1/_8$
Latency	1	1	$^1/_8$
Object Location Probability	8	8	1

value for both fast-download and streaming matrix is $\frac{0.032}{0.58} = 0.056$. The *CR* value for quick-find search is $\frac{0.0}{0.58} = 0$.

4.2.5 Measuring QoS Metrics

As discusses in section 4.2.4, there is local decision at every hop to decide where to forward the query. This decision is based on an aggregate-cost whose computation involves the parameters link bandwidth, link latency and the object location probability. The measurement and scaling of these parameters is discussed in this section.

4.2.5.1 Overlay-Link Bandwidth

Every node regularly measures bandwidths of the links connecting to its neighbours. The link at the overlay layer actually consists of many links at the Internet level and most of the time crossing the administrative boundaries as shown in the figure 4.2. It is not expected that every peer will run services that help other peers in measuring bandwidth. Therefore measurement has to be done without expecting any cooperation from the receiving peer.

Bandwidth of a link is defined as number of bits that can be transmitted over a period of time. If an overlay link consists of m physical links in the Internet each link L_i having a capacity (bandwidth) of C_i then the bottleneck-bandwidth C is defined as

$$C \equiv \min_{i=1...m}\{C_i\}\,. \tag{4.9}$$

This is the maximum bandwidth the overlay link can provide for a flow if there is no other traffic on that path. Available bandwidth $B_t$is defined as

$$B_t \equiv \min_{i=1...m}\{C_i[1-u_t]\} \tag{4.10}$$

Here u_t refers to the average link utilization rate at time t. Available-bandwidth is the maximum bandwidth an overlay link can provide for a fresh flow at time t. Bottleneck-bandwidth is fixed for a path but available bandwidth varies with traffic. Therefore it is easier to measure bottleneck-bandwidth compared to available-bandwidth of an overlay link.

The tool used in [Saroiu *et al.* 2002a] for measuring bottleneck bandwidth is Sprobe [Sariou *et al.*]. Sprobe has the capability to measure upstream and downstream bandwidths without expecting the cooperation of the target node. Recently [Jain & Dovrolis 2003] proposed that in-band bandwidth can be measured without separate endeavour. They used the data-flow itself to piggy-back certain information to measure in-band (achievable TCP throughput) bandwidth.

Once bandwidth is measured it is important to scale it. This scale should be uniform for all parameters and across the network. Such a condition will enable the requester node to compare the path-costs of different paths and chose the least-cost. To have the uniform scaling, minimum and maximum values 0, $B_{\max}$ are defined globally for the network. The range $[0, B_{\max}]$ is divided into 10 sub-ranges. Each range is given a value from 0 to 10. The ranges and the corresponding values are shown in Table 4.8. Here the bandwidth values are mapped to values in the range $0-10$ but in reverse order. That is higher bandwidths are given lower values. This is because the cost function is to be minimized. So the corresponding parameters should be given values of minimizing nature.

Table 4.8: Bandwidth ranges and normalized values- B refers to the bottleneck-bandwidth of the link

Range	Value
$\left(0, \frac{B_{\max}}{10}\right]$	10
$\left(\frac{B_{\max}}{10}, 2 * \frac{B_{\max}}{10}\right]$	9
$\left(2 * \frac{B_{\max}}{10}, 3 * \frac{B_{\max}}{10}\right]$	8
$\left(3 * \frac{B_{\max}}{10}, 4 * \frac{B_{\max}}{10}\right]$	7
$\left(4 * \frac{B_{\max}}{10}, 5 * \frac{B_{\max}}{10}\right]$	6
$\left(5 * \frac{B_{\max}}{10}, 6 * \frac{B_{\max}}{10}\right]$	5
$\left(6 * \frac{B_{\max}}{10}, 7 * \frac{B_{\max}}{10}\right]$	4
$\left(7 * \frac{B_{\max}}{10}, 8 * \frac{B_{\max}}{10}\right]$	3
$\left(8 * \frac{B_{\max}}{10}, 9 * \frac{B_{\max}}{10}\right]$	2
$\left(9 * \frac{B_{\max}}{10}, B_{\max}\right]$	1

4.2.5.2 Overlay-Link Latency Measurement

Latency is measured as the time taken for a message to travel from one end of a network to the other. Latency has impact on multimedia streaming. It is measured as Round Trip Time (RTT). RTT varies with network traffic. Therefore it is measured regularly at the overlay node. The measured RTT value is scaled down appropriately as per the Table 4.9. The large RTT values are assigned large values and vice-versa. This is to minimize the cost function. The maximum link latency across the network is taken as $L_{\max}$.

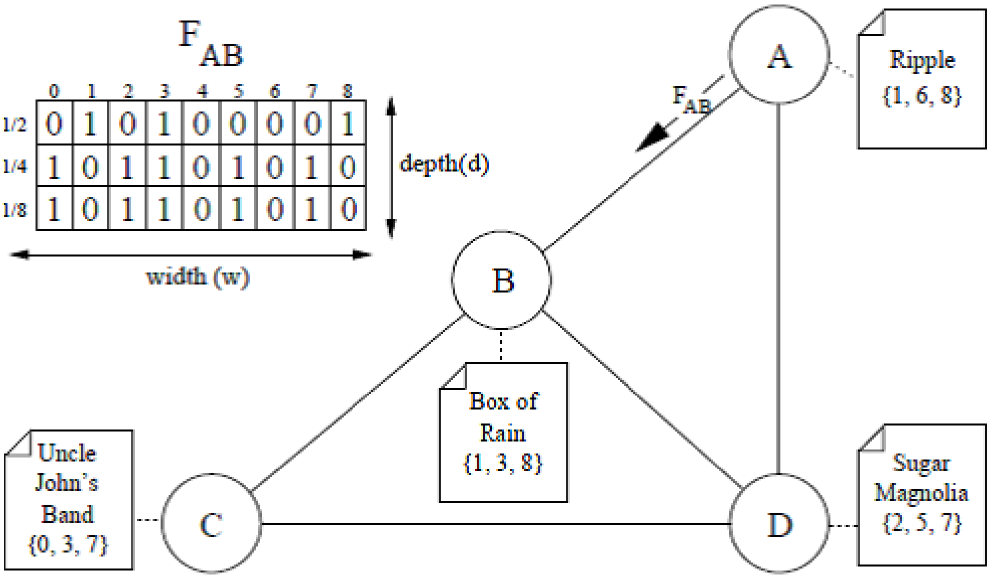
4.2.5.3 Object Location Probability

In unstructured peer-to-peer overlays the location of the object is unknown to the querying node. The query is directed blindly or intelligently at each hop to certain neighbours. The neighbours in turn forward it to others. While forwarding, the probability of finding the object in a particular direction is uncertain.

Here we have adapted a method from [Rhea & Kubiatowicz 2002], called Attenuated Bloom Filters (ABF) to give direction for the object's location. ABF is an array of Bloom Filters [Bloom 1970]. Bloom Filter is a bit vector that maps to the hash values of k hash functions that are applied over a string. It is popularly used for storing membership

Table 4.9: Latency ranges and normalized values-L refers to the latency measured over a neighbour link

Range	**Value**
$\left(0, \frac{L_{\max}}{10}\right]$	1
$\left(\frac{L_{\max}}{10}, 2 * \frac{L_{\max}}{10}\right]$	2
$\left(2 * \frac{L_{\max}}{10}, 3 * \frac{L_{\max}}{10}\right]$	3
$\left(3 * \frac{L_{\max}}{10}, 4 * \frac{L_{\max}}{10}\right]$	4
$\left(4 * \frac{L_{\max}}{10}, 5 * \frac{L_{\max}}{10}\right]$	5
$\left(5 * \frac{L_{\max}}{10}, 6 * \frac{L_{\max}}{10}\right]$	6
$\left(6 * \frac{L_{\max}}{10}, 7 * \frac{L_{\max}}{10}\right]$	7
$\left(7 * \frac{L_{\max}}{10}, 8 * \frac{L_{\max}}{10}\right]$	8
$\left(8 * \frac{L_{\max}}{10}, 9 * \frac{L_{\max}}{10}\right]$	9
$\left(9 * \frac{L_{\max}}{10}, L_{\max}\right]$	10



	0	1	2	3	4	5	6	7	8
1/2	0	1	0	1	0	0	0	0	1
1/4	1	0	1	1	0	1	0	1	0
1/8	1	0	1	1	0	1	0	1	0

Figure 4.14: Attenuated Bloomfilter F_{AB} at node A for neighbour B - source: [Rhea & Kubiatowicz 2002]

Table 4.10: Object location probability ranges and normalized values- P refers to the probability of finding the object over the neighbour link

Range	Value
$\left(0, \frac{P_{\max}}{10}\right]$	10
$\left(\frac{P_{\max}}{10}, 2*\frac{P_{\max}}{10}\right]$	9
$\left(2*\frac{P_{\max}}{10}, 3*\frac{P_{\max}}{10}\right]$	8
$\left(3*\frac{P_{\max}}{10}, 4*\frac{P_{\max}}{10}\right]$	7
$\left(4*\frac{P_{\max}}{10}, 5*\frac{P_{\max}}{10}\right]$	6
$\left(5*\frac{P_{\max}}{10}, 6*\frac{P_{\max}}{10}\right]$	5
$\left(6*\frac{P_{\max}}{10}, 7*\frac{P_{\max}}{10}\right]$	4
$\left(7*\frac{P_{\max}}{10}, 8*\frac{P_{\max}}{10}\right]$	3
$\left(8*\frac{P_{\max}}{10}, 9*\frac{P_{\max}}{10}\right]$	2
$\left(9*\frac{P_{\max}}{10}, P_{\max}\right]$	1

information of a group in relatively small storage space. ABF contains *BF*, an array of Bloom Filters each one corresponding to a level. The first *BF[0]* contains hashes of all the files at one of the neighbours and *BF[1]* contains the hashes of all the files of neighbour's neighbours. This is depicted in figure 4.14. Node A maintains ABF vector F_{AB} for the neighbour B. There are three levels. In the first level, the file names at node B are hashed using three hash functions. These values are 1, 3, 8. The corresponding bits are set in level-1 vector of F_{AB}. Similarly files available two hops away along the link AB, they are set in level-2. If the file is matching with level-1, the probability is assigned as $1/2$. If it is found in level-2, and level-3 the probabilities $1/4, 1/8$ are assigned respectively. If it is found in more than one level, the probabilities are summed up. For example, the file "Uncle John's Band" is found at level-2 and level-3. Then the probability of finding that item at A along the link AB will be $\frac{1}{4} + \frac{1}{8} = \frac{3}{8}$. The maximum probability (P_{max}) for any item can be $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} = \frac{7}{8}$.

The probabilities are also scaled to uniform values. More the probability more will be the chance of finding that item. The probability values are divided into 10 sub-ranges. They are assigned values similar to bandwidth as shown in Table 4.10.

4.2.6 Link-cost Functions

At each hop, the parameters need to be combined into an aggregate-metric. That function which combines these parameters either linearly or non-linearly is the link-cost function. They are discussed below.

4.2.6.1 Weighted Average

This is a linear combination of the metrics. At node n the link-cost for neighbour j is defined as

$$cost_j = w_{bw} \times B_j^{normalized} + w_{lt} \times L_j^{normalized} + w_{olp} \times QP_j^{normalized} \tag{4.11}$$

The weights are indicated by $w_{bw}, w_{lt}, w_{olp}$for bandwidth, latency and object-location probability respectively. Among all the links connecting to neighbours, the link with the lowest cost $cost_j$ is considered as the best alternative.

4.2.6.2 Compromise Programming

Compromise Programming defines the best alternative as the one whose point is at the least distance from an ideal point [Zeleny 1982]. The goal is to minimize the distance between the ideal f^* and actual value. We choose an alternative that is as close as possible to ideal conditions. At node n, the link-cost for neighbour j is defined as

$$\begin{aligned} cost_j \quad = \quad & \left[(w_{bw})^p \left| \frac{f_{bw}^* - B_j}{B_n^{\max} - B_n^{\min}} \right|^p \right]^{\frac{1}{p}} + \left[(w_{lt})^p \left| \frac{f_{lt}^* - L_j}{L_n^{\max} - L_n^{\min}} \right|^p \right]^{\frac{1}{p}} \\ & + \left[\left(w_{olp} \right)^p \left| \frac{f_{olp}^* - P_j}{P_n^{\max} - P_n^{\min}} \right|^p \right]^{\frac{1}{p}} \end{aligned} \tag{4.12}$$

where f_{bw}^* refers to the ideal value of bandwidth parameter among the given neighbour links. Same is applicable to other parameters. The weights are indicated by $w_{bw}, w_{lt}, w_{olp}$for bandwidth, latency and object-location probability respectively. $B_n^{\max}$ and $B_n^{\min}$ refers to the maximum and minimum bandwidth values among the neighbour links. The same is applicable to other parameters. Here, normalized values are not used. normalization is inbuilt into the equation 4.12. Signs are assigned to the parameters depending on how they contribute to the minimization of the function. Latency is given negative sign and the other two are given positive sign. p refers to the balancing factor with respect to the compensations between deviations. For $p = 1$, all deviations from f^* are taken into account in direct proportion to their magnitudes. In this work, $p = 1$ is used. After computing the cost values for neighbour links, the neighbour with least link-cost value is considered the best alternative.

4.2.6.3 TOPSIS

TOPSIS is expanded as Technique for Order Preference by Similarity to an Ideal Solution. It identifies the alternative that has shortest distance from ideal solution f^* and farthest distance from the negative ideal solution f^{**} [Chen & Hwang 1992]. For each neighbour link j, separation distance measures $D_j^+$and D_j^- are computed sing Euclidean-distance formula as shown in equations 4.13 and 4.14.

$$D_j^+ = \sqrt{\left(B_j^{normalized} - f_{bw}^*\right)^2 + \left(L_j^{normalized} - f_{lt}^*\right)^2 + \left(P_j^{normalized} - f_{olp}^*\right)^2} \tag{4.13}$$

$$D_j^- = \sqrt{\left(B_j^{normalized} - f_{bw}^{**}\right)^2 + \left(L_j^{normalized} - f_{lt}^{**}\right)^2 + \left(P_j^{normalized} - f_{olp}^{**}\right)^2} \tag{4.14}$$

f_{bw}^* and f_{bw}^{**} are the maximum and minimum bandwidths available among the neighbour links. The same is applicable to object location probability parameter. In case of latency, f_{lt}^* and f_{lt}^{**} are the minimum and maximum latency values among the neighbour links. The relative closeness to ideal solution is measure as shown in equation 4.15.

$$cost_j = \frac{D_j^+}{D_j^+ + D_j^-} \tag{4.15}$$

Among all the $cost_j$ values, neighbour with least cost value is considered as the best alternative.

4.3 Experimental Analysis

The simulations for the proposed solution are carried out on a custom-built simulator on Java platform. It was discussed in section 3.2.3.1.

Table 4.11: Simulation parameters and their values-ABF refers to attenuated bllomfilter

Parameter	**Value**
Nodes	1000
Topology	Fully Decentralized
Average Degree	9.3
Objects	150
Replicas	5000
No of queries	3008
Fast Download Search Preset	[0.73, 0.08, 0.19]
Multimedia Streaming Search Preset	[0.05, 0.76, 0.18]
Quick Find Preset	[0.1, 0.1, 0.8]
ABF Levels	3
Bloom Filter Length	253 bytes

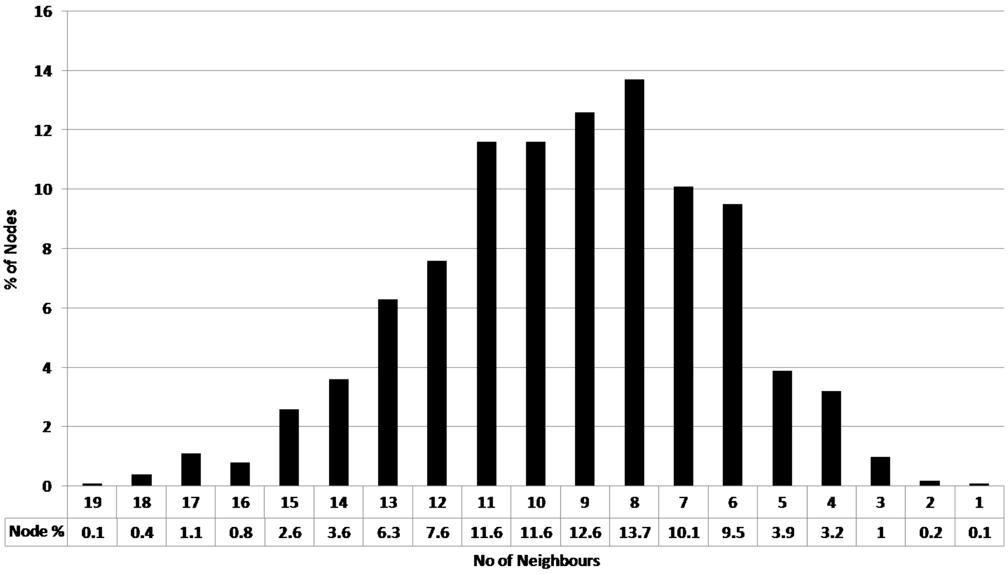
4.3.1 Simulation Setup

The simulation parameters are summarized in Table 4.11. The purpose of simulation is to measure the effectiveness of the proposed QoS search algorithm in meeting the user expectations. User priorities are derived from the network requirements for the expected service. The simulation is setup in the following steps.

- building topology from a pre-defined file
- assigning bandwidth and latencies to links
- replicating files in the network
- query generation

4.3.1.1 Building Topology

The network consists of 1000 nodes. The connectivity is according to the random model generated by GT-ITM [Zegura *et al.* 1996] topology generator. The average node degree is 9.34. The distribution of node-degree is shown in figure 4.15. The nodes are arranged in fully-decentralized fashion.



	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Node %	0.1	0.4	1.1	0.8	2.6	3.6	6.3	7.6	11.6	11.6	12.6	13.7	10.1	9.5	3.9	3.2	1	0.2	0.1

Figure 4.15: Node-degree distribution in the simulation topology - y-axis is the percent of nodes with number of neighbours projected on x-axis

4.3.1.2 Assigning Link-level Parameters

The bandwidth and latency properties of the overlay links are randomly assigned in a reasonable range. All bandwidths are taken in mega bytes per second (Mbps) and all latency values are taken to be in milli-seconds. As it can be observed from figure 4.16, there are all types bandwidths in the network. The bandwidth ranges from 1 to 99 Mbps. Similarly latency values are also assigned randomly to the links. The link-latency values range from 1 to 999 milli-seconds. The distribution of these values is shown in the figure 4.17. The distribution is uniform across the ranges.

4.3.1.3 Object Replication

There are 150 unique objects. They are replicated in the overlay according to their popularity. 1^{st} object is the most popular and 150^{th} object being the least. The objects and queries are distributed according to the observations made in [Chu *et al.* 2002]. The replica distribution of these objects is done according to the Zipfian distribution with parameter $\alpha = 0.82$. The replicas are placed at randomly selected nodes. The replica distribution is as shown in figure 4.18. The graph shows that it is a power-law distribution.

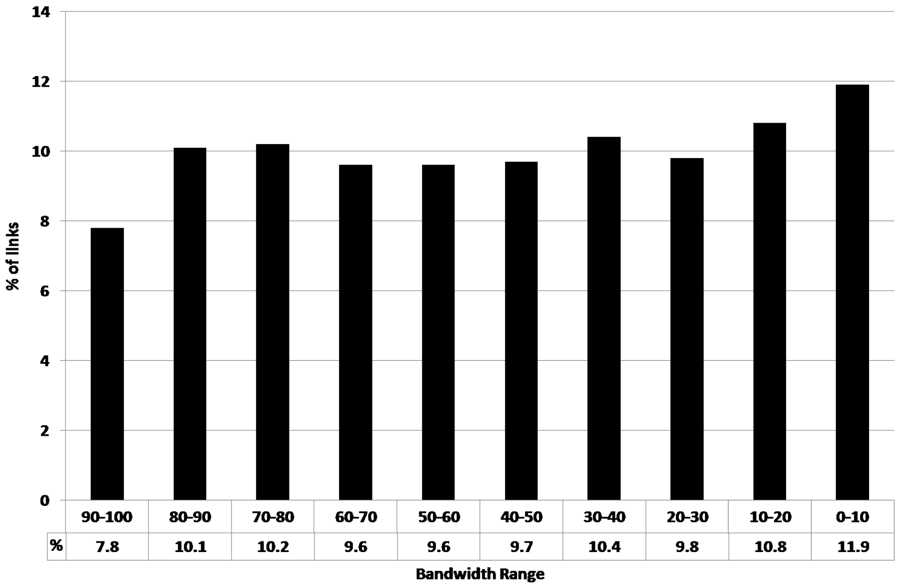


Figure 4.16: Bandwidth distribution of the overlay links in the simulation - all ranges of bandwidths are almost evenly distributed

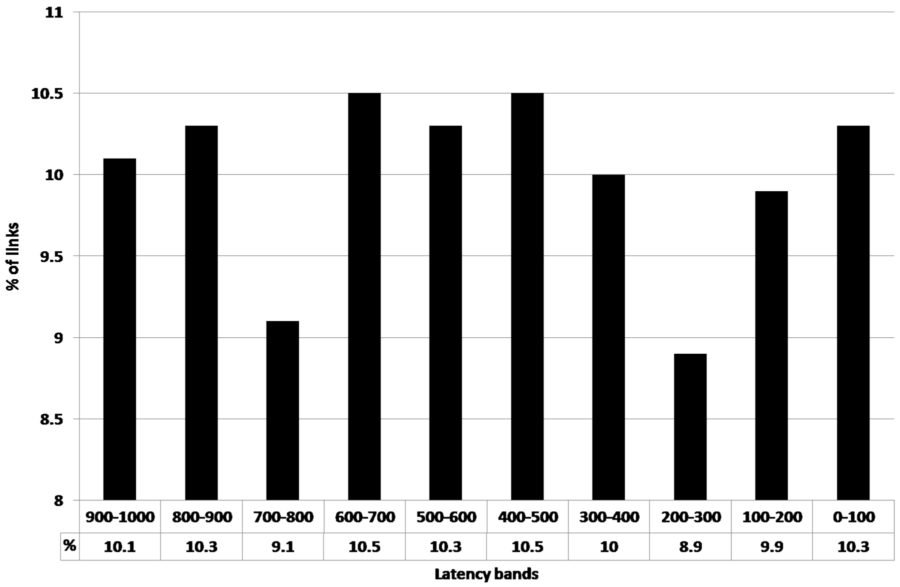


Figure 4.17: Latency characteristics of the overlay links in the simulation - all ranges of latencies are approximately evenly distributed

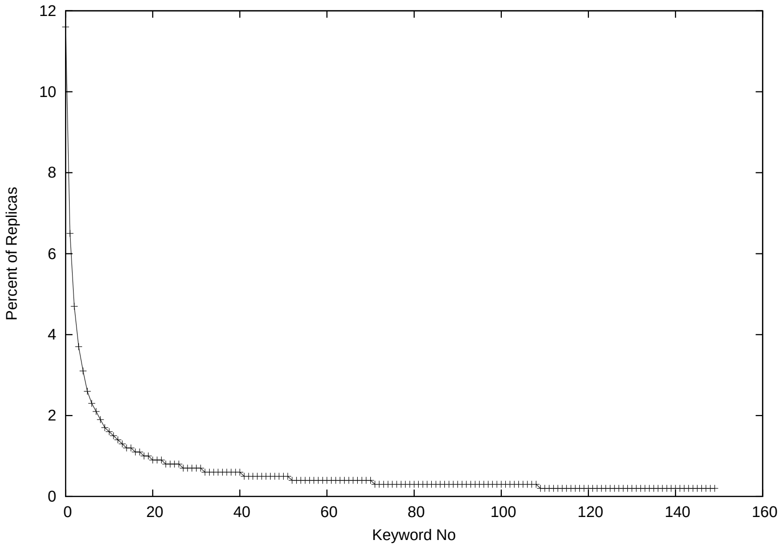


Figure 4.18: Object replica distribution in the simulation - replicas are distributed according to Zipf distribution

4.3.1.4 Query Generation

Queries for these objects follow the Zpifian distribution (α = 0.82) i.e. the popular objects receive more queries than that of less popular objects. The queries are generated according to this pattern. Query is fired every 100 milli-seconds. The query message is added to the queue of the node from which it needs to be fired.

4.3.2 Result Analysis

The purpose of the analysis is to show that the QoS-constrained search gives better results than usual methods like flooding and random walk and also fulfills the expectations of the user.

The comparison of bandwidth, latency and pathlength characteristics of QueryHit messages with respect to three search methods, namely flooding, random walk and QoS-constrained TOPSIS is explained below. Figure 4.19 shows the comparison of bottleneck-bandwidths returned by the QueryHit messages. The TOPSIS-based search returns 88.7% of the QueryHits having bandwidth above 50 Mbps whereas flooding and random walk return only 25% of the QueryHits having bandwidth above 50 Mbps.

Figure 4.20 depicts the comparison of pathlengths of QueryHit messages. Pathlength

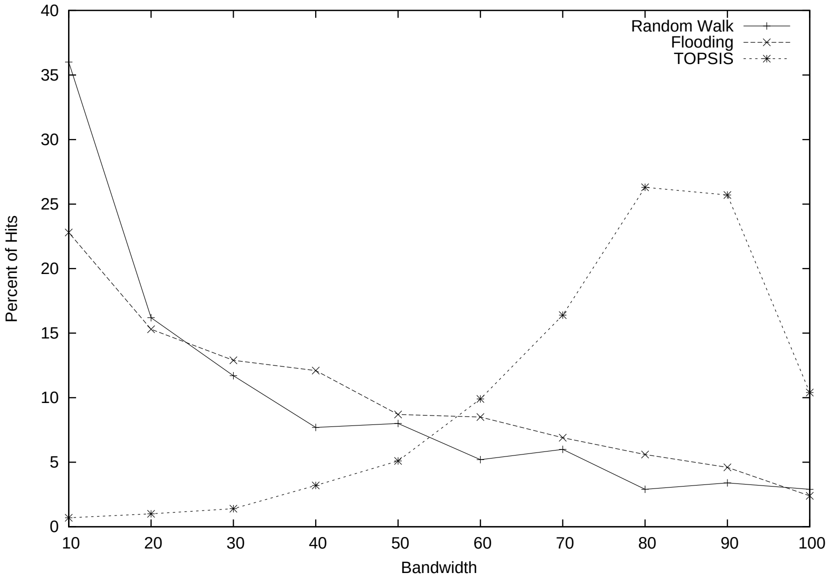


Figure 4.19: Comparison of bottleneck bandwidths of QoS-constrained search with Flooding, and Randomwalk - in QoS search, for each query, bottleneck bandwidth is measured for least-cost QueryHit

is defined as the number of hops between requesting node and the QueryHit node. The figure shows that 97% of the QueryHit messages returned through TOPSIS have path lengths between 1 and 3. Flooding has only 83% of QueryHits having pathlengths in that range. On the other hand, random walk has QueryHits with pathlengths ranging from 1 to 22. This is expected from randomwalk algorithm because of its depth-first-search nature. Randomwalk has only 62% of QueryHits having pathlengths between 1 and 3.

The comparison of accumulated delays is depicted in figure 4.21. The figure shows that 91% of the QueryHits obtained through QoS-based TOPSIS have accumulated delays less than 1000 milli-seconds. Flooding and randomwalk algorithms obtain only 54% and 42% of QueryHits having accumulated delays less than 1000 milli-seconds.

There are three cost functions used for enabling QoS-constrained search. They are Weighted Average, Compromise Programming, and TOPSIS. Bandwidth, accumulated-delay and pathlength characteristics of QueryHits obtained through these methods are compared herein. Figure 4.22 depicts the comparison of bottle-neck bandwidths. Obviously TOPSIS outperforms the other two. 62% of the QueryHits obtained through TOP-

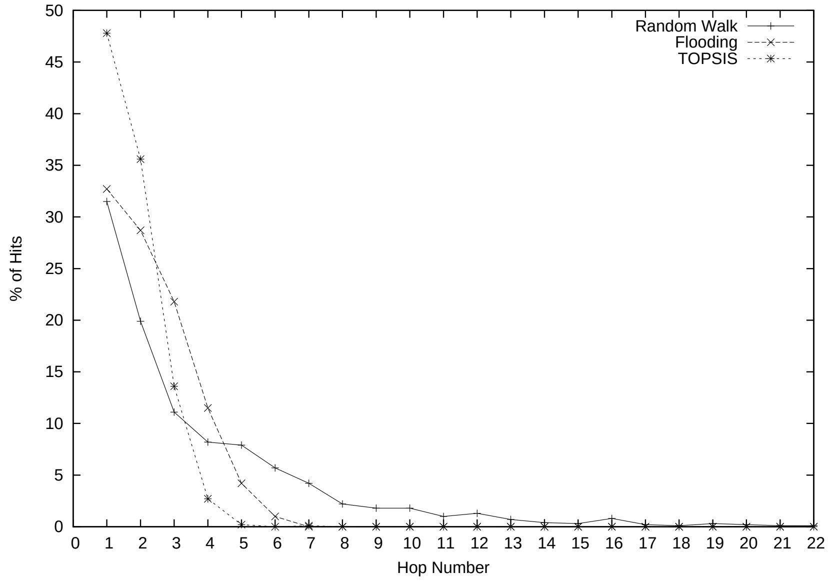


Figure 4.20: Comparison of pathlengths of QoS-constrained search with Flooding, and Randomwalk - in QoS search, for each query, pathlength is measured for least-cost QueryHit

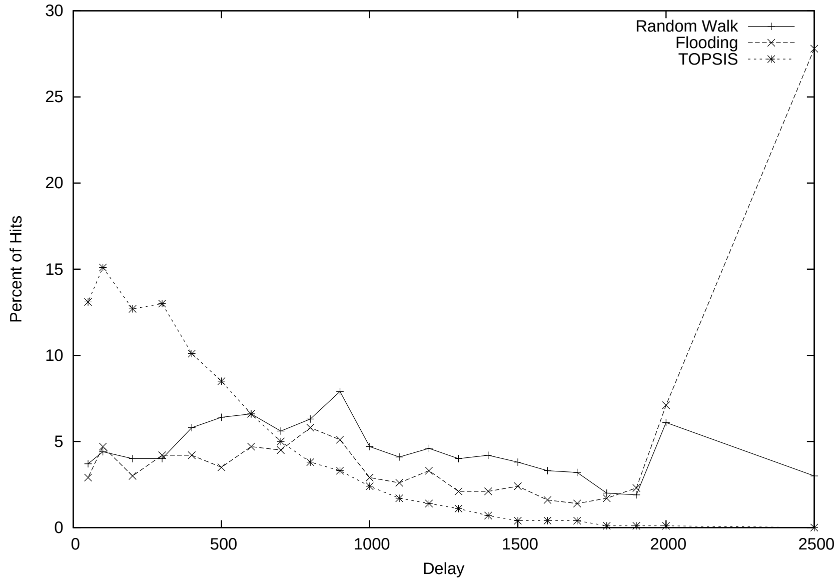


Figure 4.21: Comparison of accumulated delays of QoS-constrained search with Flooding, and Randomwalk - in QoS search, for each query, accumulated delay is measured for least-cost QueryHit

SIS are having bandwidth above 70 Mbps whereas only 52% of the QueryHits are having bandwidth above 70 Mbps in the other two methods. It can also be observed that both weighted average and compromise programming methods perform in similar ways.

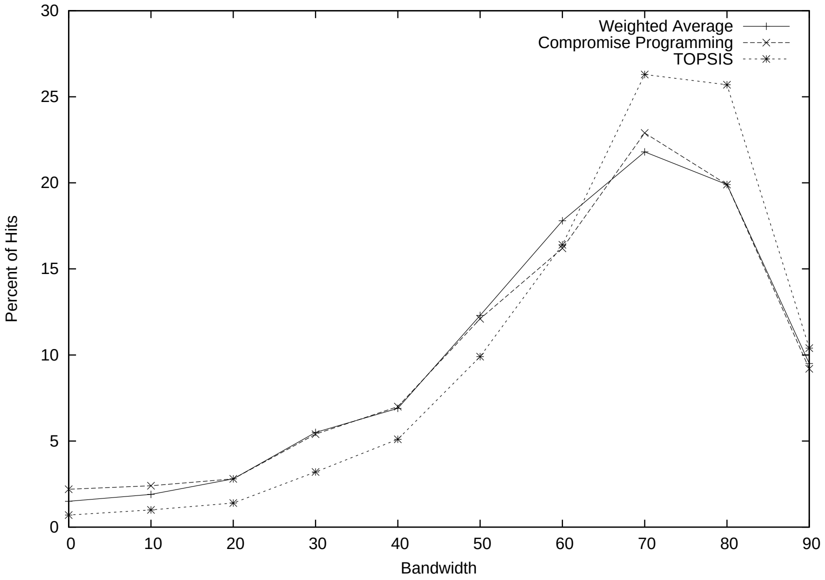


Figure 4.22: Comparison of bottleneck bandwidths by cost function - for each query, bottleneck bandwidth is considered for least-cost QueryHit

It can be observed from figure 4.23 that all three cost functions perform similarly. This is because the probability of finding the object is computed from the ABF and this factor has no global relevance. Bottleneck-bandwidth and accumulated-delays are computed end-to-end.

Figure 4.24 depicts the accumulated delays for all three methods. Clearly TOPSIS performs better than the other two. TOPSIS obtains 54% of QueryHits having delays from 0-300 milli-seconds where as the other two obtain only that of 50%. Both weighted average and compromise programming perform nearly same but towards the higher delays compromise programming gives better results.

The simulation is carried out with different preset priorities for bandwidth, delay and probability of finding the object. The same settings (topology, queries) were employed while simulating each preset priorities. The presets considered are fast-download search, multimedia-streaming search, quick-find search and equal-importance. The maximum

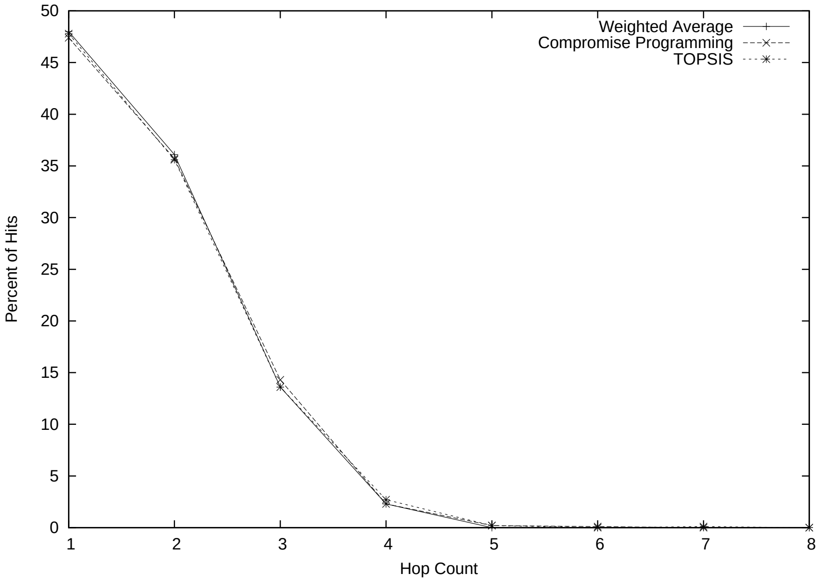


Figure 4.23: Comparison of pathlengths by cost function - for each query, bottleneck bandwidth is considered for least-cost QueryHit

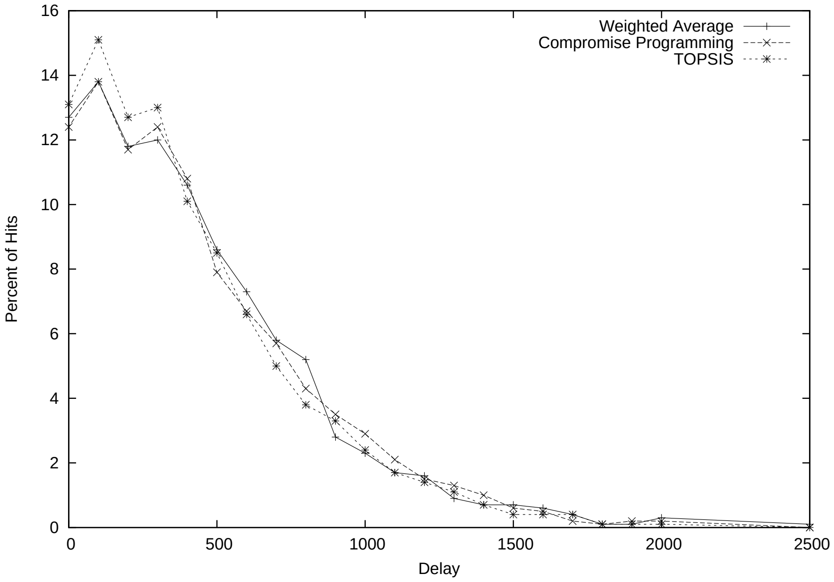


Figure 4.24: Comparison of accumulated delay by cost function - for each query, accumulated delay is considered for least-cost QueryHit

priority is given to bandwidth, delay and probability in each of these presets respectively. Equal-importance preset assigns equal priority to all three parameters. Figure 4.25 depicts the bottleneck-bandwidth comparisons of QueryHits obtained in each preset. Paths discovered with Fast-download preset, carry bottle-neck-bandwidths in higher range. This surpasses the bandwidths obtained in other preset conditions. Equal importance and quick-find preset conditions give similar results due to the equal priority given to bandwidth and latency. In multimedia-streaming preset, we see a reverse pattern of fast-download preset. This is because multimedia-streaming gives lowest importance to bandwidth.

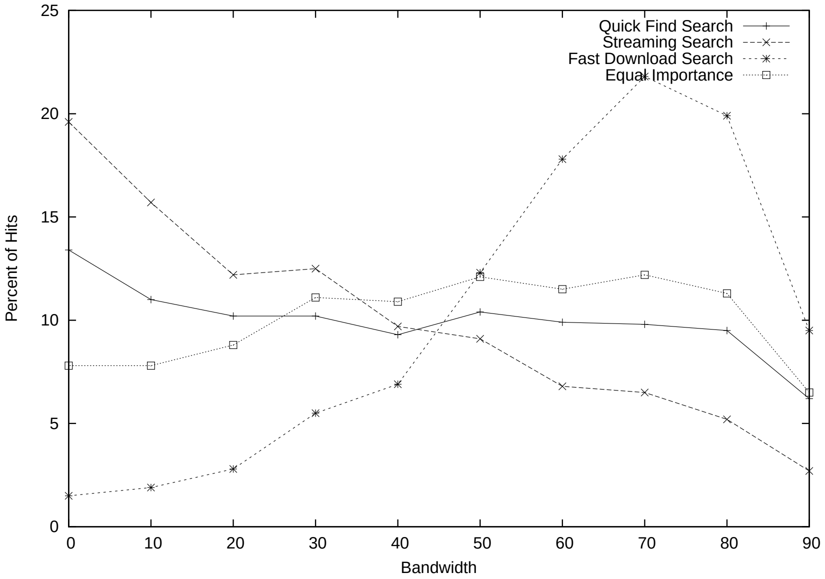


Figure 4.25: Preset-wise comparison of bottleneck bandwidths - for each query, bottleneck bandwidth is considered for least-cost QueryHit

In figure 4.26, the pathlengths comparison is depicted. Quick-find search gives the best performance in finding shortest-pathlength paths. This is the expected outcome of this preset. It should find the object quickly. Pathlength is indication of quickness of response i.e. smaller path lengths will give quicker responses. The streaming-preset and download-preset perform similarly because both treat probability of finding the object with equal priority.

Figure 4.27 depicts the comparison of accumulated delays of paths obtained in dif-

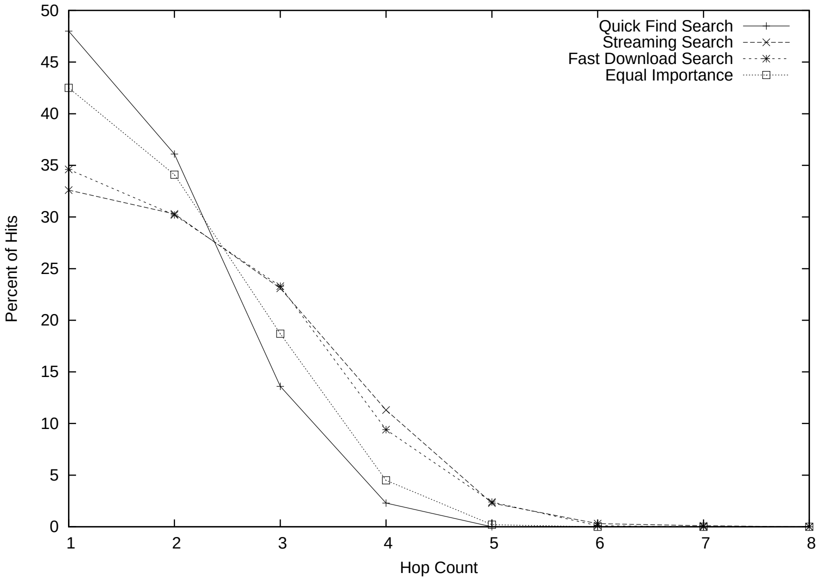


Figure 4.26: Preset-wise comparison of pathelngths - for each query, pathlength is considered for least-cost QueryHit

ferent presets. Clearly multimedia-streaming preset obtains most paths with minimum accumulated delays. Fast-download preset performs the worst because of its lowest preference to latency. But fast-download also performs as good as the flooding or random walk would have performed. The outcome expected from multimedia-streaming preset is to obtain paths with lowest accumulated delays. It can be observed that this expectation is achieved.

It is usual for a query to receive multiple QueryHits from different replicas of the object. Each QueryHit will have its own path characteristics. So far the analysis is about least-cost paths. That is every QueryHit carries a cost value accumulated over its path. Only those QueryHits having least cost are chosen. Now least-cost path characteristics are compared with paths of other QueryHits. Figure 4.28 depicts the bottleneck-bandwidth characteristics. Average bandwidth is calculated as average of bottleneck-bandwidths associated with all QueryHits received. The graph is plotted over the queries sorted by their average bandwidth. One thing to notice is that the variation of average bandwidth curve in download-preset, streaming-preset and quick-find-preset. In download-preset, the average bandwidth rises very steeply, indicating that all paths have bottleneck-bandwidths

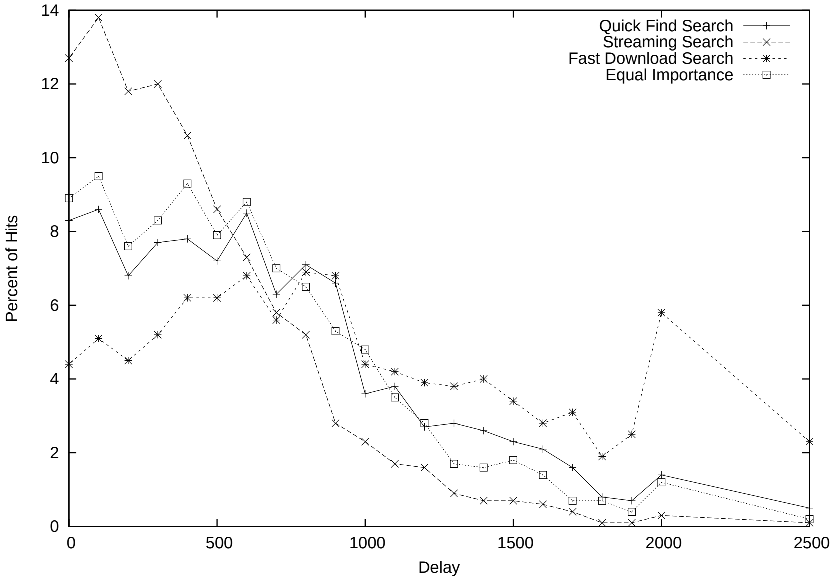


Figure 4.27: Preset-wise comparison of accumulated delays - for each query, accumulated-delay is considered for least-cost QueryHit

much above other presets. The points on either side of the average-line indicate the bandwidth of least-cost path. In most of the cases, the least-cost path's bandwidth is higher than the average bandwidth. In few cases (13%), the non-least-cost paths had better bandwidths. This can be explained as follows: the cost is computed by cost function which takes three parameters as input. The cost is added at every hop to compute end-to-end path-cost. For fast-download preset, the maximum priority is given to bandwidth. Bottleneck-bandwidth is a min-parameter i.e. the minimum bandwidth in the end-to-end path becomes the bandwidth of the whole path. Since the way end-to-end cost is computed and the way end-to-end bandwidth is computed differ, the least-cost-path need not necessarily indicate the highest bandwidth. In the case of delay, the ways of computing end-to-end delay and end-to-end cost match. Least-cost-path necessarily means minimum end-to-end delay.

In figure 4.29, both least-cost-path's accumulated delay and the average of accumulated delays over all the QueryHits received is depicted. This is done for three presets, namely fast-download search, multimedia streaming search and quick-find search. Firstly, it can be observed that the slopes of average curves are distinctly different across the pre-

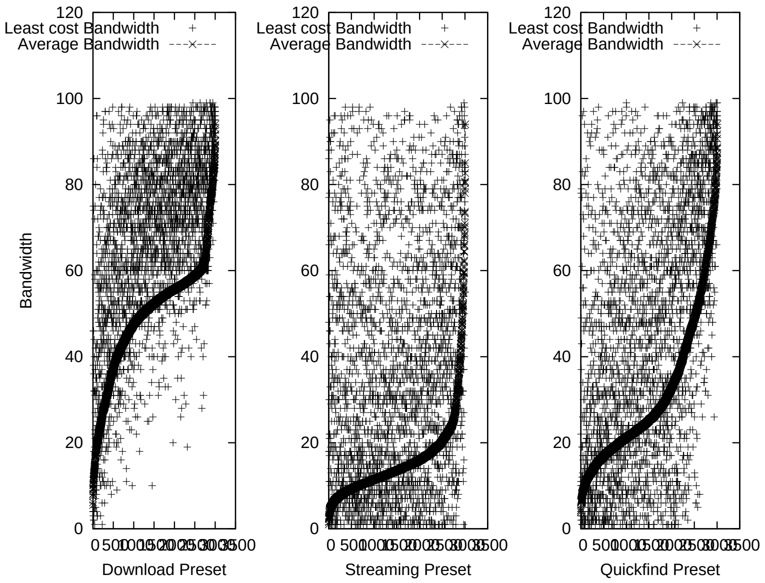


Figure 4.28: Bottleneck bandwidth comparison for least-cost path and other paths - bottleneck bandwidth obtained for least-cost path is compared to average bottleneck bandwidth of all QueryHit-paths

sets. The steepest average-curve is in fast-download preset. The slowest growing curve is in multimedia-streaming preset. For most of the queries, the accumulated-delay falls below 2000. Secondly, accumulated-delays of least-cost paths fall well below the average curve. This indicates that least-cost is an appropriate measure for finding the best QueryHit. As discussed above, the way of computing end-to-end cost and end-to-end delay are same. Therefore they have direct correspondence.

Figure 4.30 depicts the pathlengths measured of least-cost paths and the average over all QueryHit paths. The graph is plotted by sorting the queries by their average pathlength. Firstly, it can be observed that quick-find preset has the slowest growing average curve. It indicates that all the QueryHits received are pathlength aware. The least-cost pathlengths are well below the corresponding average pathlength. The least-cost pathlengths form parallel lines due to the discrete values of hops. Pathlengths can take values 1,2,3,4 only. In quick-find preset 19% of the queries have average pathlength as 1 where as in other presets it is only 0.02%.

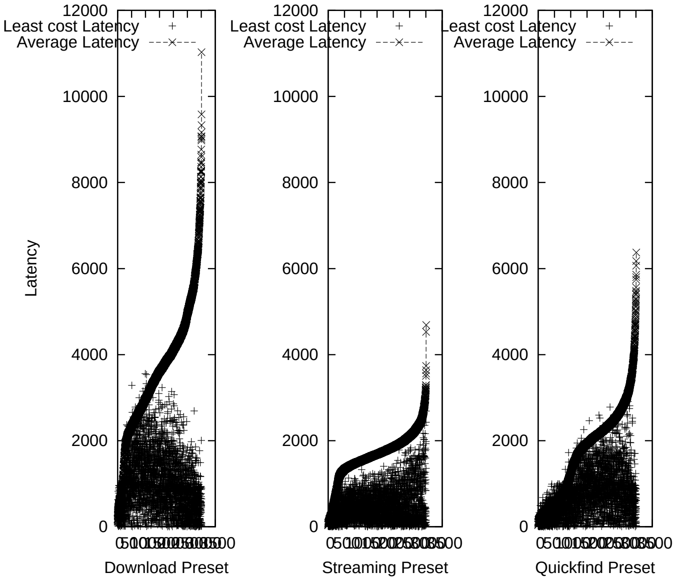


Figure 4.29: Accumulated delay comparison for least-cost path and other paths - accumulated delay obtained for least-cost path is compared to average accumulated delay of all QueryHit-paths

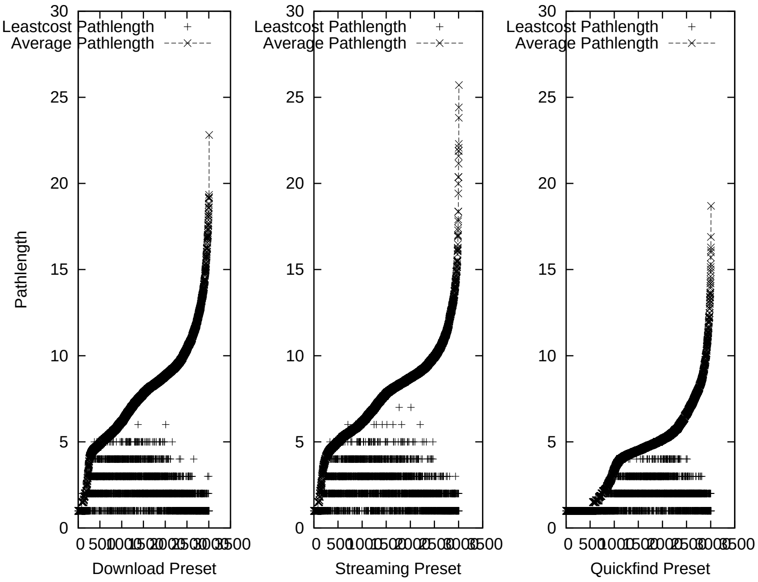


Figure 4.30: Pathlength comparison for least-cost path and other paths - path length obtained for least-cost path is compared to average pathlength of all QueryHit-paths

4.3.2.1 Comparing Results with SAMCRA

TAMCRA(or SAMCRA) proposed by [Neve & Mieghem 2000] uses a non-linear aggregate cost function. It uses this cost metric to compute feasible paths by an extended Dijkstra's algorithm. This algorithm works with the assumption that global topology with network parameters like bandwidth and latency is available at one place. In the first step, it prunes the whole network and removes the links that are less than the required minimum bandwidth constraint. Then it uses Dijkstra's algorithm to select paths with minimum cost aiming at minimizing the end-to-end delay and hops. The implementation of this algorithm is available at [sam 2006]. We have run this algorithm on our simulation topology and computed the paths for the queries used in our simulation. The topology and link parameters bandwidth, and latency are common to both approaches. In SAMCRA the bandwidth constraint is specified as 0 Mbps.

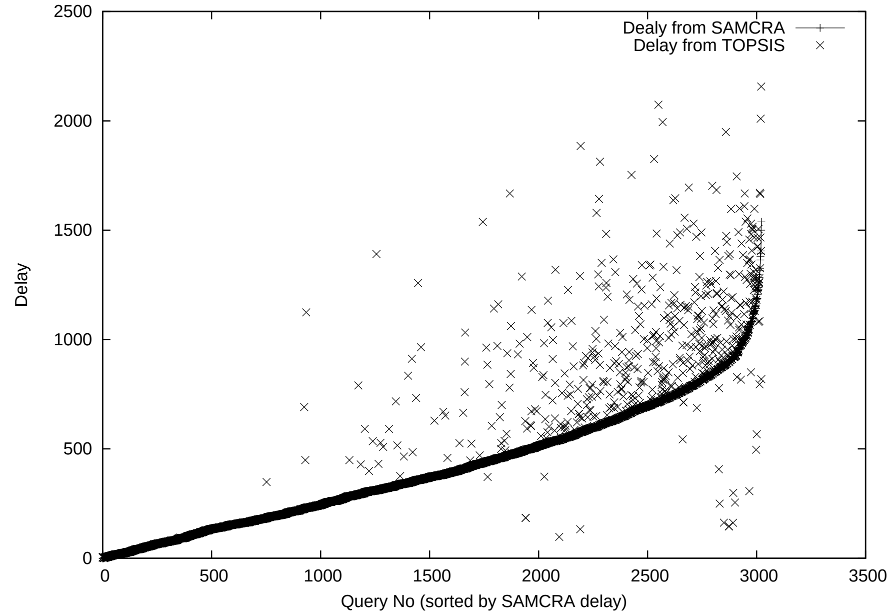


Figure 4.31: Comparison of delays obtained by SAMCRA and QoS algorithm with cost function TOPSIS - the entries on x-axis are query numbers sorted on SAMCRA delays

In our approach, streaming preset has settings with highest priority for obtaining minimum delay paths. So we compared results of SAMCRA with results obtained from streaming preset. Cost function used is TOPSIS. Delays obtained from running SAMCRA algorithm and delays obtained by running our approach are compared. The delays are plotted in figure 4.31. We found that for 82.53% of the queries, the delays are the same as that of SAMCRA results. In 16.51% of the queries, the delays are more than SAMCRA delays. In 0.96% of the cases, the delays are lower than the SAMCRA delays. The delay difference is plotted in figure 4.32. It can be observed that the difference is zero for most of the queries.

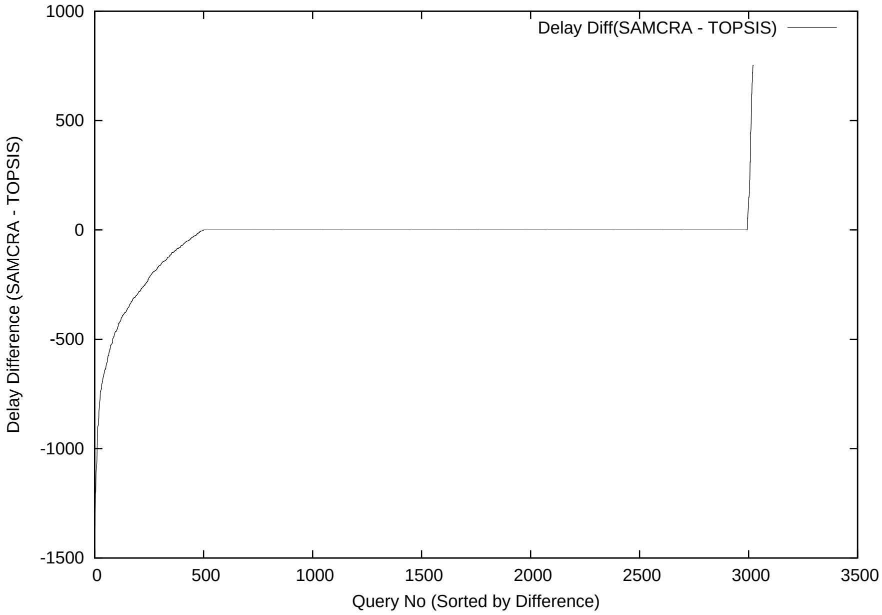


Figure 4.32: Difference of delays obtained by SAMCRA and QoS algorithm with cost function TOPSIS - delay differences are sorted in ascending order

SAMCRA also aims at minimizing the pathlengths. The path lengths obtained from SAMCRA and pathlengths obtained from streaming preset are compared in 4.33. We found that 84.95% of the queries obtained the same pathlengths in both the algorithms. In 11.97% of the queries, the pathlengths obtained through SAMCRA found to be longer. In 3.08% of the queries the pathlengths of SAMCRA found to be shorter.

Bandwidths obtained through SAMCRA and our approach are depicted in 4.34. In 84.49% of the queries, bandwidths obtained by SAMCRA and our approach are the same. In 5.52% of the queries, bandwidths obtained by SAMCRA are higher. In 10.98% of the queries, bandwidths obtained by SAMCRA are lower. Overall, QoS-constrained search algorithm has near-match performance with SAMCRA.

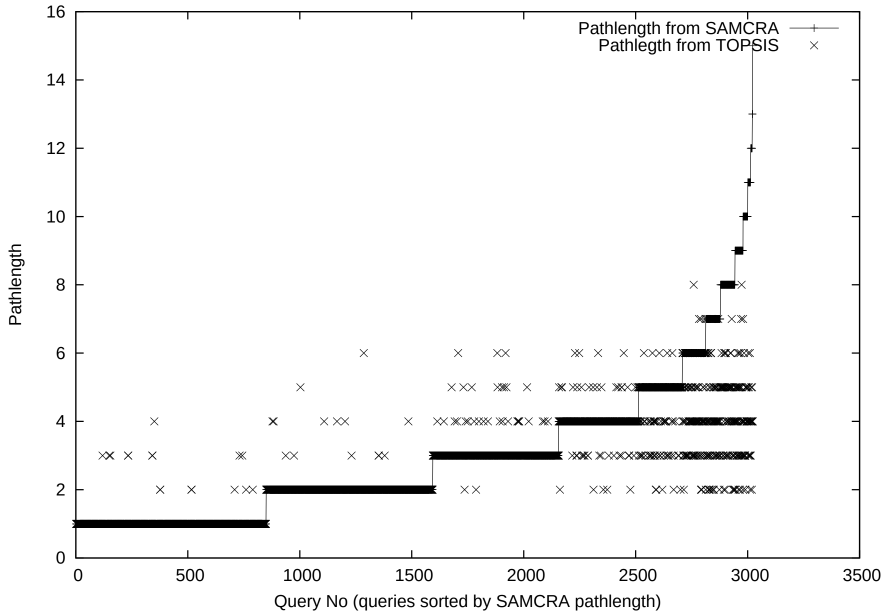


Figure 4.33: Comparison of pathlengths obtained by SAMCRA and by QoS algorithm - the entries on x-axis are query numbers sorted on SAMCRA pathlengths

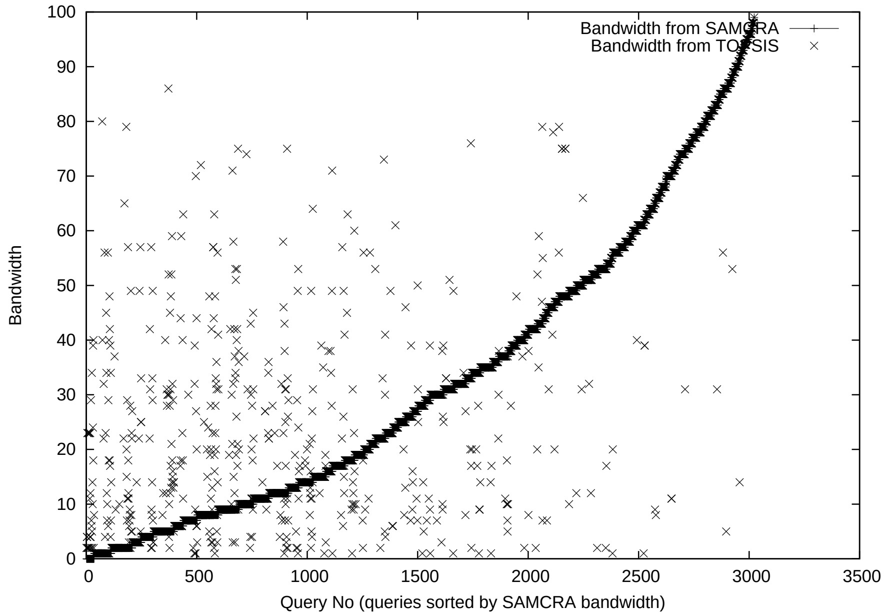


Figure 4.34: Comparison of bandwidths obtained by SAMCRA and by QoS algorithm - the entries on x-axis are query numbers sorted on SAMCRA bandwidths

4.4 Conclusion

There are several ways on how to make search efficient in peer-to-peer file sharing networks. Herein, we have attempted to address this issue by differentiating user requests and correspondingly adapting the search procedure. The proposed search algorithm is evaluated on 1000-node network. Results of simulation indicate 40-55% gains over traditional methods like flooding and random-walk. It has near-match performance (82% exact matches) with SAMCRA. The following are the advantages of the proposed search algorithm:

- It takes user preferences, machine settings into account and incorporates them into search.
- It differentiates between the requests as fast-download search, multimedia-streaming search, and quick-find search. It defines presets for them which can be picked by the user.
- Applies heuristics at each hop in order to find the best path to destination satisfying user preferences.
- It returns several replies with path costs, allowing the user to choose the path with the least cost.

The limitations of the model are:

- It doesn't guarantee that user preferences will be fully met.
- It doesn't provide a solution for reserving any resources at the nodes.

Chapter 5

Mechanisms for Detecting Sybils

5.1 Introduction

Security is a fundamental issue to be addressed when the system involves multiple users and their shared resources. Large-scale peer-to-peer overlays involve millions of user identities and their devices contributing to the functioning of the network. These distributed networks are attractive platform for malicious intents due to their decentralized nature, lack of central control, huge shared resources etc. Sybil attack is an attack intended at gaining control over a network through forged identities for misutilising resources and influencing global decisions of the network.

5.2 Related Work

Solutions for identifying or detecting Sybil attack proposed in the literature are analyzed by their methods. This is presented in table 5.1. There are three types of solutions:

5.2.1 Admission Control

In this type of solution, the identities are verified before admitting into the network. Most of the solutions are of this kind. In centralized solutions, the identities and their characteristics are maintained at a central location. They are matched with new identity to check the duplication. In distributed mechanisms, the verification is distributed among honest identities.

5.2.2 Detection

In this type of solution, the system looks for Sybil identities in the network. The detection can be by tests such as resource-challenging [Borisov 2006] or network characteristics [Wang *et al.* 2005] or personality characteristics. The solution proposed by [Wang *et al.* 2005] aims at detecting groups and then issuing computation puzzles.

Table 5.1: Sybil solutions and their methods

Solution	**Execution level**	**Means**	**Level of Defense**
Douceur 2002	Central Authority	-	Admission Control
Borisov 2006	Distributed	Computational puzzles	Detecting
Rowaihy et al. 2007	Hierarchical	Computational puzzles	Admission Control/Limiting Damage
Bazzi and Konjevod 2005	Central Authority	Geometric distance certificates	Admission Control
Wang et al. 2005	Centralized /Distributed	Netprint: router IP + MAC+ vector of RTT measurements from land mark nodes	Detection of Groups
Dinger and Hartenstein 2006	Distributed	Registering IP at r nodes	Admission Control
Castro et al. 2003	Central Authority	Signed certificates, fee per registration	Admission Control
Danezis et al. 2005	Distributed	Distributing queries among trusted nodes	Limiting Damage
Yu et al. 2006	Distributed	Trust relationships	Admission Control
Jyothi and Janakiram 2009	Distributed	Past transactional history	Limiting Damage

5.2.3 Limiting Damage

Some solutions are proposed to limit the damage caused by the Sybil identities. The damage can be corrupting the files, generating spurious replies etc.

In this chapter, we propose two approaches for Sybil attack. They are storage constrained challenge-response model, and using psychometric tests. They are discussed below.

5.3 Storage Constrained Challenge-Response Model

File replication in P2P has many advantages such as reducing traffic congestion; increasing object availability and fault tolerance. Single node failures, like crashes of nodes, can be tolerated because of the redundancy introduced by replicas. If a host of a replica fails, requester may access another host with a replica. Data replicated at more than one site also helps in efficient access. But, large scale Peer to Peer systems face security threats from faulty or hostile elements. Peer-to-Peer (P2P) based file sharing applications have become highly popular in today's Internet due to the spread of platforms such as Napster, Gnutella, KaZaa, eDonkey, BitTorrent, and others. The Sybil attack in which a single user can pose as multiple identities is a serious threat to P2P file sharing systems because the malicious entity can sabotage the replication mechanism.

Sybil attack has its impact on file sharing systems especially in replication systems. By knowing the mechanism of replication which is used in a particular P2P network, a malicious user (Sybil attacker) can create fake identities in the network so that the file replication of a particular file happens entirely or partially on the Sybil identities created by this particular user. Once the replica is in the hands of Sybil identity, it can corrupt, hide or destroy the copy. Worst case is when all copies are replicated on Sybil identities only. Sybil attack goes against maintaining quality and accessibility of content, and robustness of the network. Here we propose a challenge-response model for limiting the Sybil growth in the network.

5.3.1 System Model

The network is a structured network like Chord [Stoica *et al.* 2001] where the nodes are placed in pre-determined positions according to the node id. Node ids are generated using the SHA-1 hashing function. The Chord network is based on underlying Distributed Hash Table (DHT). The topology of Chord is ring shaped i.e. the ids are arranged in a ring shape and most of the operations are in clock-wise direction. The data is placed in the node whose id is the closest to the hashed key of the object. In most structured networks, the objects are replicated in *r* number of successors, r being dependent on individual system. The node where the object is originally stored is called ′owner′ of the object. The owner replicates the copies of the object in *r* successors. The owner of the file has details about to which nodes the file has been replicated. It is assumed that for a normal node, there will not be a situation where it doesn't have space to store the file. This is because the object placement is done by the consistent hashing [Karger *et al.* 1997]. This will ensure equal distribution of load. So no node at any time can refuse to store the file.

5.3.1.1 Threat Model

- **Peer Model:** Network consists of honest and Sybil identities. Honest identities are created by honest users and they always adhere to the protocol of the network. Sybil identities are generated by a malicious user in a large proportion. These identities may subvert the protocol for selfish goals. There can be several malicious users or Sybil entities which create Sybil identities in a large proportion. Sybil identities appear to be normal nodes but they don't have their own computational, memory and storage resources but use the resources of the Sybil entity. Identities created by one Sybil entity share the resources available with entity. For the nodes in the network, these identities appear to be non different from normal nodes. Sybil entities, through their managed-Sybil identities, influence voting objects, and use disproportionate amount of resources in their favour.

  Nodes, before joining the network, need to generate a public key/private key pair and hash the public key using SHA-1 to get its identity. This will prevent the nodes from choosing advantageous position in the topology.

- **Replication Model:** Owner of a file replicates the file among a set *R* of nodes. Nodes or identities[1] in *R* may not be Sybil identities, some may be Sybil identities or all be Sybil identities. File owner can be honest node or Sybil node. It is considered that owner Sybil identities don't carry out DOS attacks by sending repeated replica requests or replica verification requests.

[1] In this discussion node and identity are interchangeably used.

5.3.2 Proposed Solution

5.3.2.1 Storage Constraint

All the Sybil identities created by one malicious user share the same storage. So, the data that has been replicated in Sybil identities created by that malicious user will be stored in a single storage area. The malicious user has a limited storage capacity. It is assumed that it is not economically viable for the malicious user to scale the storage in proportion to the number of Sybil identities. This model is depicted in figure 5.1. Nodes 3, 5, 6, and 8 are Sybil identities. These nodes don't have their own storage because they are virtual and are actually running in a single system. The owner of file1, file2 and file3 is node 1. When it replicates these files, the copies that go to Sybil identities are all stored in a single storage space at Sybil entity 5.

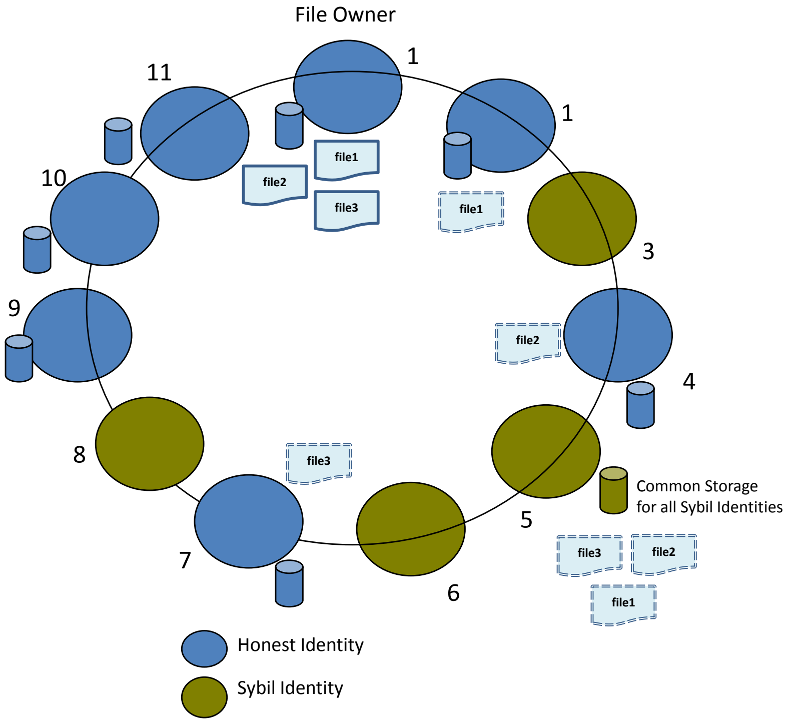


Figure 5.1: System model for storage-constrained challenge-response Sybil detection - the number of replicas to be stored by a Sybil user are proportional to the number of Sybil identities created by him

5.3.2.2 Algorithms for Detecting Sybil Identities

The following algorithms outline how the Sybil detection test happens.

- **File Owner:** The owner is the node in which DHT has stored the file. Thus every node in the network receives a set of files F. The owner node replicates every file in the set F on a set of successor nodes R. How to select this set of successors is predefined globally. How many replicas to make for each file in F is a global constant ($REPLICA\_LIMIT$) in the network. The owner node keeps track of the nodes in which it has replicated each of the files in F. However the file owner checks whether the successor is a Sybil and avoids storing the replicas in that node. Replication procedure is shown in figure 5.2.

```
Replicate(File:$f$)
begin
    SuccessorsList: $R$;
    SybilDetectedList: $D$;
    Int: $noReplica = 0$;
    ReplicaList: $L$;
    foreach successor $s$ in $R$ do
        if $s$ in not in $D$ then
            /* putIntoNetwork() utilizes the DHT framework to
            store the file                                    */
            $putIntoNetwork(s, f)$;
            add the pair $(s, f)$ to $L$;
            $noReplica = noReplica + 1$;
            if $noReplica > REPLICA\_LIMIT$ then
                break from for loop;
            end
        end
    end
end
```

Figure 5.2: Procedure for replicating a file f by file owner

Owner node verifies the existence of the replicas for all files in the set F at regular intervals. The verification is done by requesting a random sequence of bytes from the replica owner. The verification message consists of *{fileId, fromByteOffset, toByteOffset}*. The file owner sends a message to each node where the replica is placed asking for a randomly chosen byte range within the file. After sending the verification request, the owner waits for the reply. The receiver is expected to send the verification reply which contains those few bytes extracted from the object and within the expected time. The verification reply message consists of *{fileId, fromByteOffset, toByteOffset, bytes}*. When the file owner receives a reply, it verifies the byte range. If either the reply is not received or the reply is received and is not correct then, the owner notes the node identifier of the node. If the same situation occurs for more than $THRESHOLD$ number of times, the owner suspects the node to be a Sybil identity. Then it will not replicate the objects any more on this node. The suspected node is added to Sybil set S. The replica verification algorithm is shown in figure 5.3. The procedures $makeVerificationMessage()$ and $verifyReply()$ used in figure 5.3 are shown in figure 5.4 and 5.5 respectively.

```
VerifyReplications(ReplicaList:$L$)
begin
    SybilDetectedList: $S$;
    ReplicaList: $L$;
    VerificationMessage: $v$;
    VerificationReplyMessage:$vr$;
    foreach replica $r$ in $L$ do
        $v \leftarrow$ makeVerificationMessage($r$);
        send verification message $v$ to $r.s$;
        wait for $VERIFICATION\_TIMEOUT$;
        $vr \leftarrow receiveReply()$;
        if verifyReply($r, v, vr$) $= false$ or $vr = null$ then
            $L.noReplyCount \leftarrow L.noReplyCount + 1$;
            if $L.noReplyCount > THRESHOLD$ then
                add $r.s$ to S;
            end
        end
    end
end
```

Figure 5.3: Procedure for verification of a replica of file f by file owner

- **Replica Owner:** Replica owner is the node in which a file owner has requested a replica to be placed. It is the responsibility of the replica owner to store it and supply whole or part of the file when requested by file owner. For a normal honest node, the number of replicas it has to store will be within the storage capacity. It never gets a problem of shortage of space because the network ensures that no node gets more than what it is supposed to store. This is ensured by using consistent

```
VerificationMessage $makeVerificationMessage(Replica : r)$
begin
    Int: $fromByte$;
    Int: $toByte$;
    VerificationMessage: $v$;
    $fromByte \leftarrow \frac{getRandomNo()}{sizeof(r.f)}$;
    $toByte \leftarrow \frac{getRandomNo()}{sizeof(r.f)}$;
    if $fromByte > toByte$ then
        swap $fromByte$ and $toByte$;
    end
    $v.f \leftarrow r.f$;
    $v.fromByte \leftarrow fromByte$;
    $v.toByte \leftarrow toByte$;
    return $v$;
end
```

Figure 5.4: Procedure for making a verification message by file owner

```
$verifyReply(Replica : r, VerificationMessage : v, VerificationReplyMessage : vr)$
begin
    actualData ← byterange of $r.f$ from $v.fromByte$ to $v.toByte$;
    if actualData matches vr.data then
        return true;
    end
    return false;
end
```

Figure 5.5: Procedure for verifying a reply sent from a replica owner by file owner

hashing [Karger *et al.* 1997]. In case of Sybil identities, since all the Sybil identities are sharing the same storage space, soon they will run out of storage space. Thus the Sybil identities can neither say no to storing the replica nor have space to store that replica. They simply drop it or find some malicious means to store it. When file owner sends a verification message, Sybil can't answer it or answers it wrongly. If this happens consistently, the file owner detects it to be a Sybil identity.

The proposed solution detects Sybil identities in a decentralized way and also protects replicas from losses. The detection happens not at one time but over the life time of the network. This behaviour is depicted in the simulation section. The replica losses are because they stored on Sybil identities. Once the network identifies which nodes are Sybil identities, then gradually the nodes avoid storing in those Sybil identities. Thus the losses reduce in proportion to Sybil detection. This behaviour is also proved in simulation studies.

5.3.3 Attack Strategies of an Adversary

The various ways in which a Sybil Node can attack or disrupt the functioning of the file replication are given below:

- **Drop Replica:** A Sybil identity may drop the replica. This is detected by the verification process outlined before.

- **Store Replica Partially:** A Sybil identity may store only a partial portion of a replica in order to save storage space. This is detected by the verification process outlined before.

- **Sybil Identity Subverting the Solution:** As discussed in previous section, it is possible that a Sybil identity may take to illegitimate ways of storing the replica when it is short of storage space. The Sybil, when requested by owner of an object to replicate, it might ask any other honest node to store and this way overcome the shortage of space for storing large number of objects on its disk. When the file owner sends a verification request, the Sybil identity can simply forward it to the honest node and get the reply. This way the Sybil can protect itself from being detected. This problem can be solved by looking up the hashed key of the file. The file is put into the network by hashing its contents and getting a file id. The file is stored in a node whose id is most proximate to the file id. Now when the honest node gets a request for storing a file, it can derive the file id and lookup if the node requesting is same as the node where the file is stored. This will give a definite answer whether to accept the request or not. When a Sybil identity sends a malicious request, the honest node can easily detect it.

- **Content Deletion:** Sybil identities may delete the files replicated on them. A malicious user may decide to delete all copies of a particular music file on his group of Sybil identities. This may lead to non-availability of that file although it may be a very popular one.

  In the proposed solution, file owner verifies the existence of the replica it has deposited. If it doesn't get the reply with in *THRESHOLD* number of times, it disconnects from that Sybil identity and stops storing replicas on that identity.

- **Content Concealment:** The Sybil node can possess the file and not send it to the requesting node. In this case, the Sybil identity might possess the data (so that if the owner node verifies, it would be able to respond and confirm it) and but conceal it from other requesting nodes.

  A node requests a download request to a replica owner but replica owner refuses to serve it although he has the copy. In such a situation, requesting node should do the lookup on the file id and find owner. It can inform the file owner. If the file owner receives such complaints more than THRESHOLD number of times, it can suspect that it is a Sybil identity and severe its connections with it.

- **Content Pollution:** The Sybil identities can replace all or part of the content with white noise, cutting the duration, shuffling blocks of bytes within the digital recording, inserting warnings of the illegality of file sharing in the recording, and inserting advertisements; the main aim being to render the file unusable and thereby reducing its popularity. Now, this polluted content can be replicated on a large number of honest or Sybil nodes in the P2P Network. A normal user who is oblivious to all these, downloads these content and thus the polluted content spreads throughout the file sharing network eventually exceeding the number of original copies. As the users download more and more polluted copies, it might lead to frustration among users and subsequently leading them to abandon the file sharing. For example, when a recording company is on the verge of releasing a song that will likely be popular; the rival record company might pay a pollution company to spread bogus copies of the song through one or more P2P networks thereby reducing the popularity of the file.

The solution for this problem is to limit the Sybil growth so that not a majority of replicas fall on Sybil identities.

5.3.4 Experimental Setup

Simulation was carried out on a 1000-node Chord network. We used PlanetSim [Ahulló & López 2008] overlay network simulator. Necessary changes were made in the Node classes to represent the current purpose of simulation. New procedures were written for replication and verification. The simulator was a step based simulator. Every step, the messages are transferred from current node to next node. The simulation was carried out for 45000 steps. The files are replicated in the system throughout the simulation using a Poisson process with average as 4. Each node can store 50 files. Node makes 5 replicas of each file on its successors. The threshold value for terming a node as Sybil is 4. The waiting time for a verification reply is set to 10 seconds. The topology of the Chord network is shown in 5.6.

In the beginning of the simulation, all the honest and Sybil nodes are created. The honest nodes are 1000 in number. The Sybil nodes are varied from 50 to 850 i.e. 4.7% to 46% for different experiments. Whenever the messages are transferred from one node to another, whenever a replica request is made, replica is placed or failed to place in a node, in all such events necessary information is recorded in log files. These log files are later used to analyse the behaviour of the network. The results are discussed in the following section.

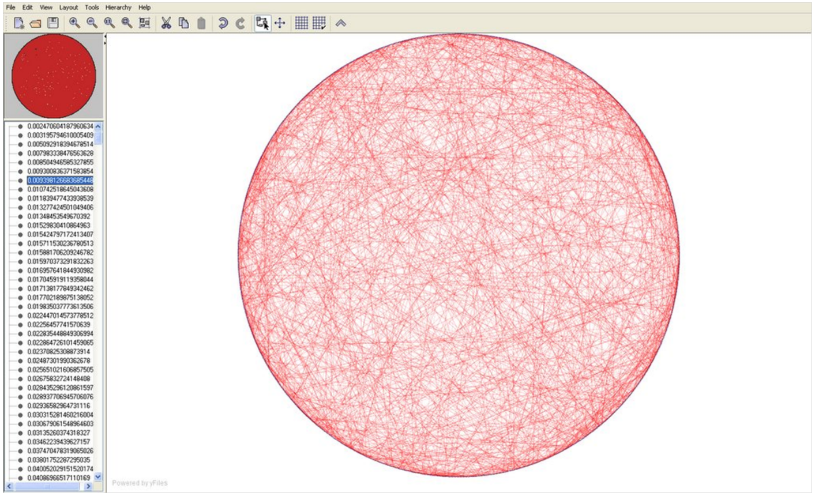


Figure 5.6: Chord network topology setup for simulation - the lines across the circle are the finger pointers

5.3.5 Results Analysis

The proposed solution could detect 84%-42% of the Sybils in 45000 steps when the Sybil proportion is varied from 5% - 46% respectively. The detection effectiveness is analysed below with respect to the parameters like initial percent of Sybil identities, total number of objects replicated in the network, waiting time for a verification reply etc.

Table 5.2: Number of Sybil nodes removed with different compositions of Sybil identities - percentage of Sybil identities is out of total identities (honest+Sybil) in the network

Honest nodes	Sybil Nodes	Sybil Identities (%)	Total Sybil Identities removed	% of Sybils removed
1000	50	4.76	44	88
1000	150	13.04	115	76.67
1000	250	20	183	73.2
1000	350	25.93	240	68.57
1000	450	31.03	300	66.67
1000	550	35.48	361	65.64
1000	650	39.39	393	60.46
1000	750	42.86	439	58.53
1000	850	45.95	374	44

The honest nodes are 1000 in number. They are kept constant for all experiments. But the Sybil nodes are varied from 50 to 850 i.e. 4.7% to 46% in different experiments. The details are shown in table 5.2. The percent of Sybil identities in the network affect the probability that a replica is placed on a Sybil node. More the percent of Sybil identities are, more is the probability that a replica is placed in a Sybil identity.

5.3.5.1 Sybil Detection Pattern

Graph in figure 5.7 shows that there is rapid detection initially and later the curves are not changing much i.e the slope is not changing drastically. As the time progresses, and as the Sybil identities are detected, the remaining Sybil identities presence in the network reduces. So the probability that a replica is placed on a Sybil identity also reduces. That is why the steepness of the curves reduces.

Also we can observe that as the proportion of the Sybil identities in the network is increased, the time taken to detect Sybil identities also increases. This is due to the fact that the number of files being stored in the network and number of their replicas being distributed kept constant. When the number of files is kept constant, and the Sybil identities proportion is increased, then the probability of placing a replica on a Sybil identity reduces. This means that if the proportion of Sybil identities is more, then the number of files replicated should also be more. This pattern is depicted in figure 5.8.

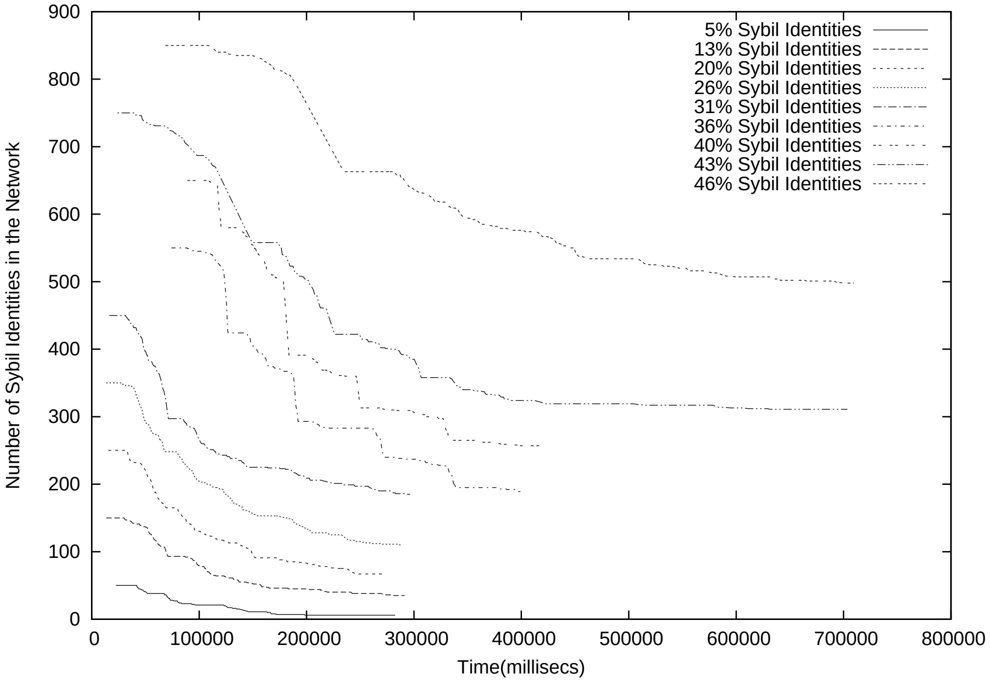


Figure 5.7: Sybil detection pattern at the different % of Sybil identities in the network - time is measured from the beginning of the simulation

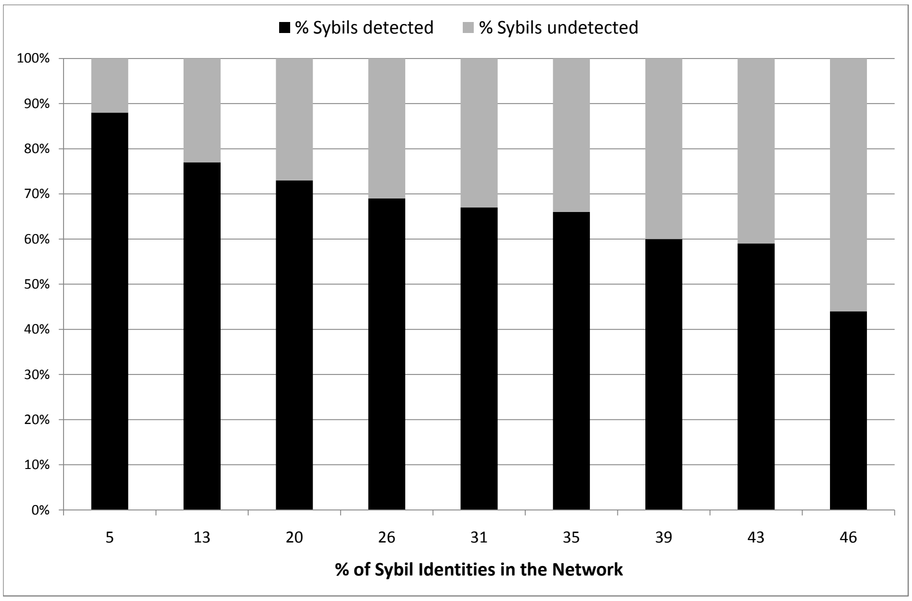


Figure 5.8: Effect of % of Sybils on detection algorithm - Network size is 1000 nodes. The data presented here is only for the first few minutes of the simulation, long enough to compare

5.3.5.2 Sybil Detection Vs Replica Losses

In figure 5.9, we see that as the Sybil identities are detected, the replica losses are also reduced. Thus reducing the number of Sybil identities has direct effect on file losses incurred in the network due to Sybil identities. We can see from the Sybil CDF that when it has reached a slow progress state, accordingly the file losses also have reduced. Normally the replica losses are due to the Sybil identities because they don't have the storage space to store replicas of all the Sybil identities. When such failures are detected, the files are replicated on a different set of nodes probably honest nodes. That way the file replicas are safer.

5.3.5.3 Effect of Number Replicas on Sybil Detection

We have already seen that detection slows down if we increase the number of sybils but keep the number of files constant. The positive side of increasing the number of files is shown in figure 5.10. Here we can see how the Sybil detection procedure is dependent on the number of files replicated in the network. The whole algorithm is dependent on the

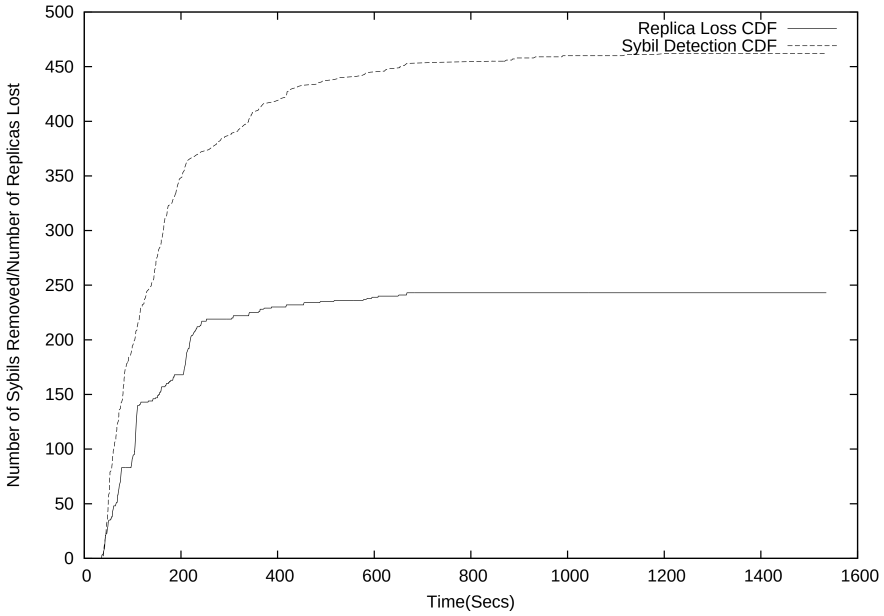


Figure 5.9: Reduction of replica losses by detecting Sybil identities - loss of replica is due to lack of storage space in a Sybil entity

replicas of files. More the number of files replicated, more is the probability that they will be replicated on Sybil identities and better will be the detection of the Sybil identities.

5.3.5.4 Effect of Waiting Time on Sybil Detection

In figure 5.11, it can be observed that, the waiting time for a verification reply from a node has no drastic influence of the detection of Sybil identities. A node may not get a reply due to reasons like the replica owner may not be present in the network, network errors, or replica owner may be on a slow network connection. There may be a possibility that a small timeout may lead to false positives. But this doesn't happen as several nodes replicate their objects on a Sybil node. The increase in waiting time doesn't delay the detection because there are several other nodes which are verifying meanwhile. So if one fails, the other may detect the node to be a Sybil identity.

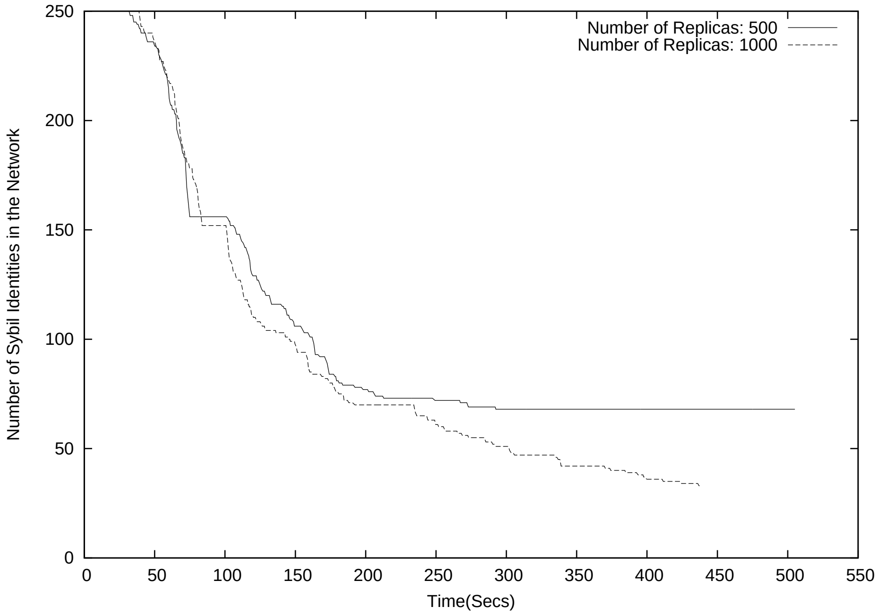


Figure 5.10: Effect of number of files replicated in the network - Number of Sybil identities detected depends on the number of objects replicated in the network

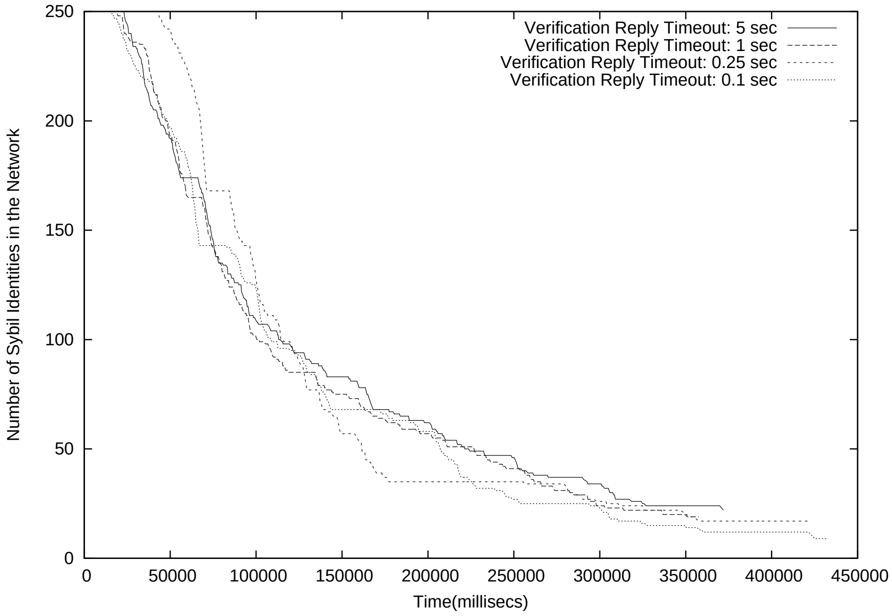


Figure 5.11: The effect of verification reply timeout on Sybil detection - the effect is almost nil

5.3.6 Conclusion

In this work we presented a novel decentralized protocol for limiting the corruptive influence of Sybil attacks on replication system in peer-to-peer networks by detecting Sybil identities and there by avoiding storing replicas on them. This approach relies on the principle that Sybil doesn't scale its storage capacity to the factor of its identities. Also unlike the other challenge-response approaches, this approach is more reliable because the storage is persistent. Here it is not difficult to simultaneously test the storage capacity of most identities because it can be done over a period of time. Experimental evaluations on this approach have shown that Sybil identities were detected to the extent of 60% of initial Sybil identities when the 40% of the nodes in the network were Sybil identities. Also the effect of parameters like initial percent of Sybil identities, total number of objects replicated in the network, waiting time for a verification reply are analysed. The simulation results show that this approach can detect Sybil identities to the degree that loss of file replicas are reduced to less than .0001%. However, this approach is less efficient when the number of replicas being maintained is less.

5.4 Detecting Sybils using Psychometric Tests

An approach for detecting Sybil Groups using Psychometric tests is proposed here. The following are the purposes of this work:

- to study the feasibility of using psychometric tests to assess the characteristics of the participants
- to devise methods to overcome some of limitations of this method
- to measure the extent of the effectiveness with which we can use this technique
- to understand the advantages of approaching Sybil groups instead of Sybil identities.

In our current solution, we aim at a partially decentralized solution that detects Sybil groups using personality characteristics.

5.4.1 Background

The solution with psychometric methods is first of its kind. In this section a brief background on the degree of difficulty of detecting Sybils in the network and applicability of psychometric tests is presented.

5.4.1.1 Detecting Sybil Groups vs Detecting Sybil Identities

The strength of Sybil attack depends on the number of identities it creates and what fraction of the network they occupy. The influence on the network is exerted as a group but not at the individual identity itself. Therefore Sybil attack is very difficult to detect at the identity level. In figure 5.12, it can be observed that there are entities and identities (blue coloured circles). Some entities have represented themselves as multiple-identities forming large fraction of the network.

Having a centralized server issuing logins or providing authentication itself is not sufficient to prevent Sybil attacks. Amazon, eBay etc have centralized authentication systems but still they face Sybil attacks. The principle behind preventing Sybil attack is that one should be able to map the real infrastructure entity to the virtual identities and then put a limit on such number of mappings. Such a system requires one to authenticate the identity by a physical proof such as photo identity card, credit card etc. Such a restriction on enrolling new entities into the system severely limits the spread of systems among users.

Sybil entities may not necessarily be creating disturbance in the network such as launching distributed DoS attacks, or dropping packets or poisoning the routing tables, partitioning the network etc. It may be difficult to say what a Sybil identity is doing is wrong. It may be doing the same thing like any other honest identities. For example, a Sybil entity can position its identities strategically at different places in the network and make sure that every packet in the network will pass through at least one of its identities so that the Sybil has control over the network. Looking at the identity, one can’t say it is doing something malicious. It should be determined looking at the network level. Another example is that a Sybil entity can increase the reputation of a particular file by making all of its identities respond positively to the reputation metric query. Looking at the individual identity it is difficult to say that what it is doing is wrong because every identity is free to respond positively or negatively to the file request.

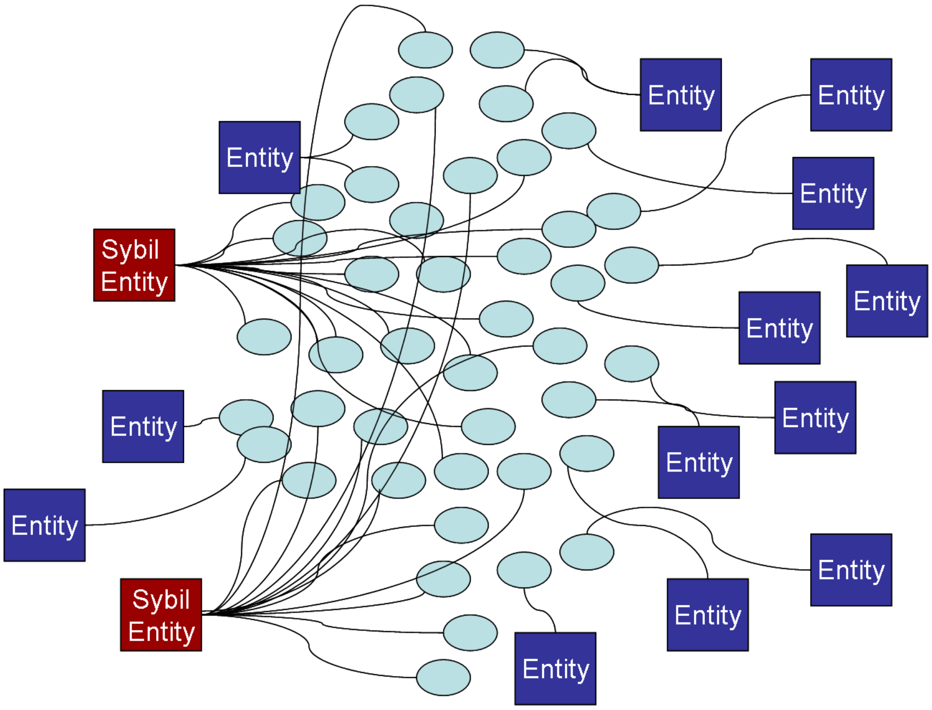


Figure 5.12: Sybil groups and Sybil identities - Sybil groups (Red colored rectangles) and Sybil identities (Blue colored circles)

So the approach to detecting Sybils in a peer-to-peer network should be free from any assumptions about the behaviour at the identity level.

5.4.1.2 Psychometric Tests

There are many theoretical approaches to conceptualizing and measuring personality. Some of them include the Minnesota Multiphasic Personality Inventory (MMPI), the Five-Factor Model [Books 2010]. There are some tools developed to measure personality. They include Personality and Preference Inventory (PAPI) and the Myers-Briggs Type Indicator (MBTI) [Books 2010]. PAPI measures personality in work environment. It is designed to determine behaviours and preferences which are related to workplace. The MBTI measurement is a psychometric questionnaire designed to measure psychological preferences in how people perceive the world and make decisions [Myers & Myers 1995]. In this paper we use MBTI due to its applicability to normal population [Pearman & Albritton 1997]. MBTI test categorises the human personality into four pairs of cognitive functional types:

- Extraversion-Introversion

- Sensing-Intuition
- Thinking-Feeling
- Judgement-Perception

These terms have specific meanings much different from those indicated by the normal usage of the term. These meanings are not relevant in this study. MBTI states that an individual has preference to one of the functions in a pair. The questionnaires are designed to reveal these preferences.

5.4.1.3 Luscher Short Color Test

[Lüscher & Scott 1971] proposed a colour test to assess the human personality. The test is based on selecting 8 colours according to individual's liking for the colours. Each colour has an objective meaning and subjective meaning. The objective meaning remains the same for all individuals. But the subjective meaning of the colour is dependent on the position of the colour in the ranking of the colours by the individual. Liking for a colour has deep connection with the psychology of the person. The colors are shown in figure 5.13. Their psychological significance is listed in 5.3.

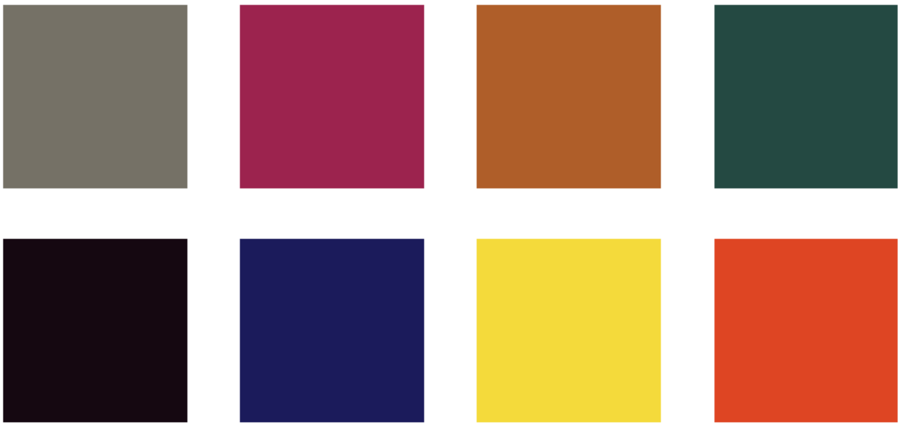


Figure 5.13: Colors in Luscher Short Color Test - each color has a psychological significance

In this study, the objective is to group the identities based on their common personality characteristics. For this purpose MBTI and Luscher colour tests are employed. Since the purpose is to find the similarity amongst the personalities of identities in the network, the

Table 5.3: Colours and their psychological significance-[Lüscher & Scott 1971]

Colour	**Traits linked to the color**
Orange-Red	Represent "force of will" and correspond to desire, domination, sexual interests, aggression, controlled passion, concern for others.
Blue-Green	Represent "elasticity of will" and corresponds to persistence, resistance to change, self-assertion, obstinacy, possessiveness, positive self-esteem, concern for self.
Dark-Blue	Represent "depth of feeling" and has emotional correspondence with tranquility, calmness, recharging, contentment, tenderness, unification, sensitivity, love and affection.
Bright-Yellow	Represent "spontaneity" and correspond to exhilaration, originality, expectancy, variability, desire to be active.
Violet	Represent intuitive and sensitive understanding of the unreal. Considered for mystical intimacy or understanding.
Brown	Represent sensation as it applies to bodily senses indicating a need for or hopeless forfeit of bodily comfort. It indicates a strong need to overcome a bad situation causing discomfort (physical or emotional).
Black	Represent absolute boundary where life ceases. It means a person rejecting and renouncing everything out of stubborn protest.
Gray	Represent intellect, knowledge and wisdom. Also implies long-lasting, classic, dignified, authoritative. Represent neutrality as it is between black and white.

interest is only in the metrics. So we limit our discussions to finding correlations rather than the actual personality of the individuals.

5.4.2 System Model

We model our solution on a super-peer type unstructured network which employs a Gnutella like protocol between client nodes. This allows for a questionnaire to be sent to the peers as a request and then, the solved questionnaire as a response.

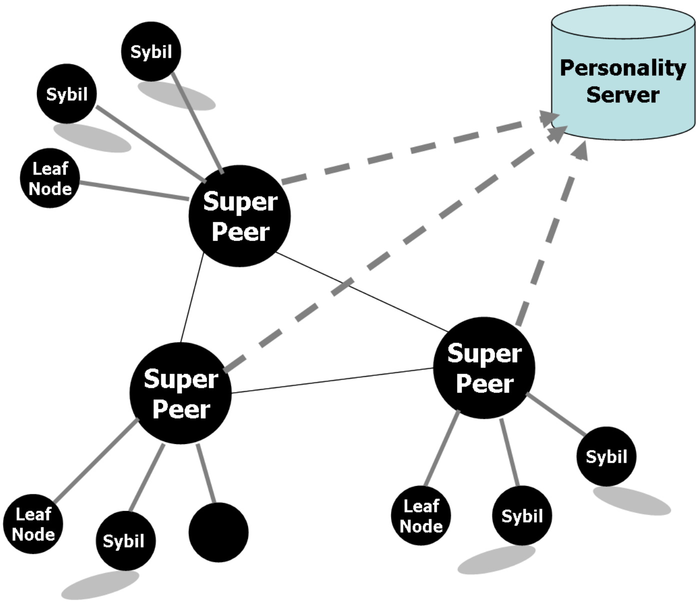


Figure 5.14: Sybil detection architecture - personality server is a centralised component

The architecture is shown in figure 5.14. The components involved are leaf nodes (some are Sybil identities), super-peers, and Personality server. Personality server is a central server that collects and stores the personality metrics of the leaf nodes. Having central server in a distributed network is not a flaw as long as that central server itself is not involved in routine network operations such as routing etc.

5.4.2.1 Threat Model

- **Peer Model:** Network consists of honest and Sybil identities. Honest identities are created by honest users and they always adhere to the protocol of the network. Honest nodes answer the questionnaires sincerely when they are sent a questionnaire by the super-peer. Sybil identities are generated by a malicious user in a large proportion. These identities may subvert the protocol for selfish goals. There can be several malicious users or Sybil entities which create Sybil identities in a large proportion. For the nodes in the network, these identities appear to be non different from normal nodes. Super peers are honest nodes. They adhere to the protocol.

- **Questionnaire Model:** A super peer may have one or more Sybil identities registered under it. Here the interaction can be of two types. Super peer interacting with honest node and super peer interacting Sybil node. Communications in these interactions are encrypted with the public keys of the receiving nodes.

5.4.3 Proposed Solution

In this section we will discuss about the proposed method.

5.4.3.1 Outline

As discussed in previous Section 5.4.1, Sybils are present in groups. Extent of their influence is proportional to their group size. There can be several Sybil groups within a peer-to-peer network. In essence a Sybil group is defined as the set of identities created by a single individual. The identities act as per the individual's want them to act. If the identities occupy a large fraction of the network, then the functioning of the network can be easily affected for selfish purposes. The solution to this problem is to cluster identities which have similar personality characteristics. Such clusters are then tested using challenge-response protocols as described in section 5.2. To identify personality characteristics, each identity in the network is given personality tests, Myers Briggs Psychometric Test and Luscher Color Test to answer. These answers are collected and analyzed at one place to identify the clusters.

5.4.3.2 Protocol

The principle communications involved in the network apart from the routine communications are 1) Super peer → Leaf node 2) Leaf Node → Super peer 3) Super peer → Personality Server 4) Personality Server → Super peer. These communications and their contents are explained below.

Leaf node's Behaviour: A leaf node receives a questionnaire from a super peer at regular times. The contents of the questionnaire are explained in the coming section. Super peer specifies the time within which it should receive the answers to the questionnaire. This is one way to differentiate between the honest and the Sybil entity. If the node doesn't answer with in the expected time although it is still active in the network, it is given a warning that it might be labelled as a malicious node. In case, the node still does not answer even after some time interval after first warning, the node is then black-listed or treated as a malicious node and taken out of the network. For the node to again come back, it has to join as a new node and follow the protocol.

Super-peer's Behaviour: Super-peer generates a questionnaire from a pool of questions corresponding to each personality trait. Then, it will send it to the leaf nodes. It will act depending on the response or no response of the leaf node. After collecting the answers it will send them to Personality Server.

Personality Server: The personality server receives the data from the super peers. It periodically runs a clustering algorithm to cluster the nodes based on their psychometric values. For every cluster discovered, it issues simultaneous computational puzzles to the nodes in that cluster. Generally the cluster contains at least some of the Sybil identities belonging to the same Sybil group. This fact is confirmed in our experiments as discussed in coming sections. In case of the presence of Sybil identities in a cluster, the Sybil entity will not be able to respond with in expected time with the answer to the resource-intensive-computational puzzle for each Sybil identity. An honest node can answer the challenge with in the expected time. By this differentiation between Sybil identities and honest nodes, the Personality server can identify all these late-responding identities within the cluster as one Sybil group. These identities are communicated to super-peers so that they may restrict their cooperation or disconnect them.

5.4.3.3 Questionnaire Preparation

The questionnaire will evaluate the psychometric index of the peers based on MBTI or Luscher Color Test. The questionnaire consists of 3 kinds of questions. Firstly, there are questions based on each of the 4 categories of personality traits according to MBTI model which gives us the information about the psychological orientation of a person.

The personality traits are extroversion-introversion, sensing- intuition, thinking-feeling, judging-perceiving. For e.g., we can ask a person whether he likes to hear less and talk more or he likes to hear more and talk less. In tables 5.4 - 5.6, some of the questions which help us to know about each of the traits in a person are presented. The options to these questions are designed in an appropriate way relating to one's work or colleagues or family so that users may spontaneously answer the questions.

Table 5.4: Sample questions for testing introvert quality-these questions enquire the psychological personality of the person

Quality	**Questions**
Extroversion-Introversion	Do you talk more than listen or vice-versa?
	Do you have high energy or quiet energy?
	Do you want to stay behind scenes or want a public role?

Table 5.5: Some questions for testing intuition quality-these questions enquire the psychological personality of the person

Quality	**Questions**
Sensing-Intuition	Do you focus on details or do you see the big picture?
	Do you work at a steady pace or bursts of energy?
	Do you trust gut instincts or actual experience?

Table 5.6: Some questions for testing feeling quality-these questions enquire the psychological personality of the person

Quality	**Questions**
Thinking-Feeling	Do you appear cool and reserved or warm and friendly?
	Do you value honesty and fairness more or harmony and compassion?
	Do you tend more to see faults or you are quick to compliment others?

Secondly, we have 2 sets of color tests (each color test has 8 colors) in each questionnaire. The leaf nodes are to fill in their preferences of colors from most preferred to least preferred. Each colour is associated with some particular trait [Lüscher & Scott 1971].

Table 5.7: Some questions for testing perceiving quality-these questions enquire the psychological personality of the person

Quality	Questions
Judging-Perceiving	Do you work first, play later or play first, work later?
	Do you like to make and stick to plans or keep flexible plans?
	Do you like freedom to be spontaneous or find comfort in schedules?

These associations are findings of the research in psychology field. For example, the orange-red color represent "force of will" & correspond to desire, domination, aggression, controlled passion, concern for others. The list is given in table 5.3.

Also, the questionnaire includes CAPTCHAS inserted at random places to prevent automated answers. CAPTCHAS are designed to be understood only by humans but not by machine.

5.4.3.4 Questionnaire Evaluation

The answers submitted by the leaf peers to super-peer are communicated to Personality server. The evaluation is carried out at the Personality server. There are two types of answers in a questionnaire. One is the set of options selected for the MBTI questionnaire. The other is list of rankings given to the 8-set colours.

To evaluate the similarity between the colour rankings of two different identities, rank-correlation coefficients are used. Kendall's rank-correlation coefficient τ and Spearman's rank-correlation coefficient ρ are used to cluster the identities based on colour test. The τ is computed by finding the concordant (N_c) and discordant pairs (N_d) for n number of items to be ranked.

$$\tau = \frac{N_c - N_d}{\frac{1}{2}n(n-1)} \tag{5.1}$$

The ρ is computed by finding the squared differences of rankings given to all n items.

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \tag{5.2}$$

The options selected by identities for MBTI questionnaire are transformed into a vector of weights. Each vector consists of 8 weights corresponding 8 qualities in four dichotomic cognitive functional pairs namely extraversion, introversion etc. There is one vector corresponding to one questionnaire. The similarity between two such vectors is found by using two metrics Cosine similarity metric and Pearson correlation coefficient. The cosine similarity metric for two vectors A and B of length n is computed as

$$\cos(\theta) = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(B_i)^2} \times \sqrt{\sum_{i=1}^{n}(A_i)^2}} \tag{5.3}$$

The Pearson correlation coefficient of two random variables X and Y with $\overline{X}$ and $\overline{Y}$ as their means is computed as

$$\frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2} \times \sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}} \tag{5.4}$$

The clustering algorithm chosen is DBSCAN [Ester *et al.* 1996]. It has the following advantages in the present context.

- It doesn't need number of clusters to be specified as input parameter.
- It supports arbitrary shaped clusters.
- It has concept of noise data and leaves out noise points.
- Cluster formation doesn't depend on the ordering of the data.
- The algorithm works well with high dimensional data.

The algorithm finds the clusters on the principle of neighbourhood of a point. Generally the neighbourhood is found with Euclidian distance. But Euclidian distance becomes an ineffective measure for high dimensional data. This problem is also known as curse of dimensionality [Bellman & Corporation 1957]. Therefore in this work, we have used the above mentioned metrics to find neighbourhood of a given point. The time-complexity of the algorithm is $O(n^2)$ and space-complexity is $O(n^2)$. By using R^*-tree indexing structures the complexity can be brought down to $O(n \log n)$.

The algorithm works on the notion of *density-reachability* and *density-connected* properties of a two points in the dataset. The algorithm takes two parameters ε and *minPoints*. ε

is the neighbourhood radius. *minPoints* is the minimum number of points required in the neighbourhood of a point to form the cluster. The algorithm is very sensitive to these two parameters. It groups all the points that fall within the radius ε of the current point and its neighbours into one cluster.

5.4.3.5 Cluster Validation

One of the most important issues in cluster analysis is the evaluation of clustering results to find the partitioning that best fits the underlying data [Halkidi *et al.* 2001]. What values for parameters ε and *minPoints* are needed so that the algorithm would give the best fit? We have used external criteria to validate clustering. Rand Statistic [Rand 1971], Jaccard Coefficient [Jaccard 1901] and Fowlkes and Mallows index [Fowlkes & Mallows 1983] are used to measure the cluster goodness.

Consider that all data points with d dimensionality are present in dataset D. Consider that $P = \{P_1, P_2, ...P_n\}$ are the predefined clusters and $C = \{C_1, C_2, ...C_n\}$ are the clusters formed out of DBSCAN algorithm. Every pair of points (x_i, x_j) in the dataset D can be assigned to one of the following sets depending on the condition it satisfies.

- True Positives (n_{00}): if both points x_i and x_j belong to the same cluster of C and to the same cluster of P

- False Positives (n_{10}): if both points x_i and x_j belong to the same cluster of C and to different clusters of P

- False Negatives (n_{01}): if points x_i and x_j belong to different clusters of C and to the same cluster of P

- True Negatives (n_{11}): if point x_i and x_j belong to different clusters of C and to different clusters of P

Rand Statistic gives ratio of true positive pairs (TP) and true negative pairs (TN) to total pairs in the data set. It is defined as

$$R = \frac{(n_{00} + n_{11})}{n_{00} + n_{10} + n_{01} + n_{11}} \tag{5.5}$$

Jaccard coefficient gives ratio of true positive pairs to total of true positive, false positive (FP) and false negative pairs (FN). It is defined as

$$J = \frac{n_{00}}{n_{00} + n_{10} + n_{01}} \tag{5.6}$$

These two indices take values in the range [0,1]. If the values are nearer to 1, closer are the two points. Fowlkes and Mallows index is computer as

$$FM = \frac{(n_{00} + n_{11})}{\sqrt{(n_{00} + n_{01}) \times (n_{10} + n_{11})}} \tag{5.7}$$

5.4.3.6 Limitations

- **False Positives:** A honest user's psychology may match with the psychology of a malicious user, thus falling in the same cluster as the malicious user. Here the honest user also has to go through the challenge-response test. This imposes an overhead on the honest user.

- **False Negatives:** The test is based on psychological metrics. So it might happen that the psychometric ratings of some Sybil identities coming from the same Sybil group may not fall within the same cluster boundaries. This way some of the Sybil identities may escape getting detected. This introduces false negatives in our proposed model. First of all such deviations of psychological pattern happen in few cases as found in our experiments. Secondly, since majority of the Sybil identities of a group are detected, the group origins can be traced and thus detect the missed out Sybil identities.

5.4.4 Attack Strategies of an Adversary

- **Not Answering:** A Sybil identity may not answer the questionnaires. In this case super-peer may penalize that node by limiting its access to resources or may disconnect it from it.

- **Random Answering:** Another apparent limitation is that what if the users give random answers by some automated means or the person himself. One way to address the automated answers is by inserting CAPTCHAs in the questionnaire. Suppose a person is answering but randomly. To address this issue, super-peer by random selection stores a copy of the questionnaire and the answers received. Super-peers use this to cross verify the answers received later. If the answers are random, then it is very less likely that answers would match. Since Sybil attacks comprise large number of Sybil identities, it is not feasible for a person to answer the questions individually for all identities.

- **Replay Attack:** A Sybil node can store a copy of the answers and may reproduce the in future. This can be answered as:

  A questionnaire consists of randomly selected questions. Each question is reframed in several other forms expecting the same answer. There are challenges like CAPTCHAS or simple questions in between. The questions and answers are randomly ordered. So users can't blindly store the answers and respond. Of course, this requires a huge question bank.

- **Direct the Questionnaire to an Honest Node:** Since the communications are encrypted with self-generated public/private key pair, a node can verify the authenticity of the super peer. If the assumption that super-peer is not an honest node is violated, then this will become a complex problem.

5.4.5 Experiment Setup

The experiment for validating our solution is carried out by conducting a survey. Survey is chosen instead of simulation because the validation involves responses of humans depending on their psychological personality characteristics. These characteristics are difficult to simulate on a machine.

The survey is conducted among the faculty and students of our university for duration of 15 days through the medium of web. Totally 185 people took part in the survey. Each of them was requested to participate more than one time. The frequency of attempts is depicted in figure 5.15. The participants are requested to attempt the survey at different times so that they are not remembering the answers. Still there can be some degree of randomization or overlaps in their responses. But overall, the survey resembles the peer-to-peer scenario where each participating identity is given a questionnaire to solve.

There are 46.49% participants who attempted the survey only once. They are considered as honest entities in the network. There are 28% percent participants who attempted the survey more than 4 times. There are 25% participants who attempted the survey more than once and less than 5 times. The participants who attempted the survey more than once are classified as Sybil entities each entity representing Sybil identities equal to the number of times they have attempted the survey. The survey is mapped to a network of 577 identities; out of which there are 86 honest identities and rest all are Sybil identities grouped in 99 groups. The maximum Sybil identities are 21 in one group, the minimum and the average being 2 and 7.6. The Sybil groups can be classified into two classes:

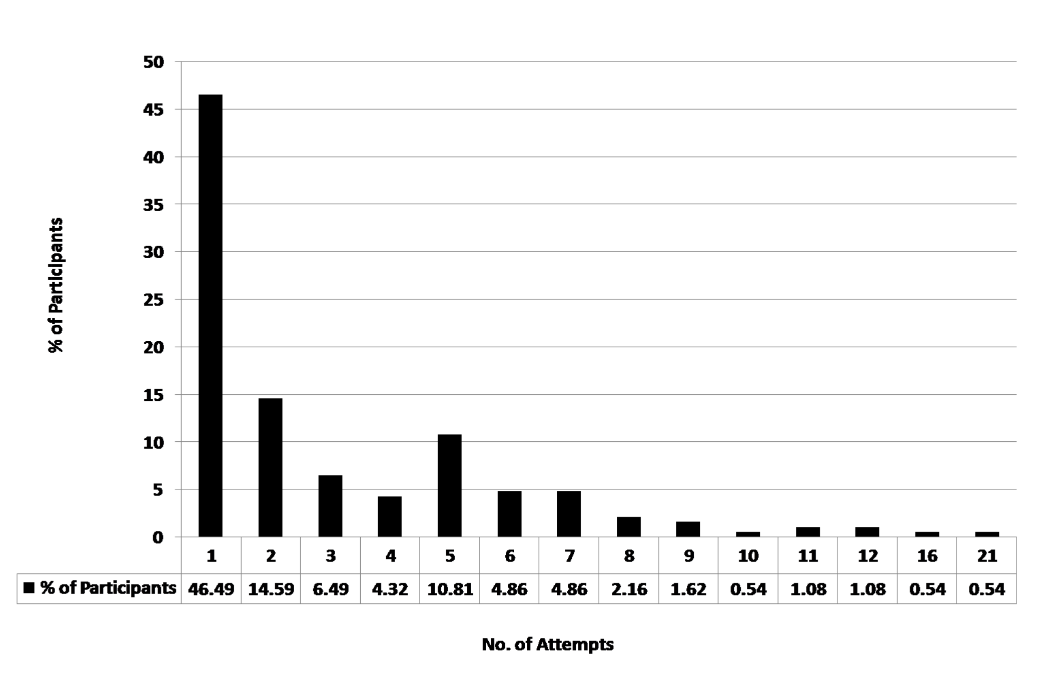
- **Weak Sybil groups**: The weak Sybil groups are those whose number of identities is less than 5. There are 47 weak Sybil groups observed.
- **Strong Sybil groups**: The strong Sybil groups are those whose number of identities is equal or greater than 5. There are 52 strong groups observed.

The Sybil groups in the mapped network are depicted in figure 5.16.

Figure 5.17 shows the standard deviation of responses by individuals along with the number of times each have attempted. 65% of individuals or Sybil groups have standard deviation more than 0.2 indicating that some of their responses are outliers i.e. they are too different from other responses. Those Sybil groups with standard deviation less than 0.2 mostly have 2 identities indicating high probability of being the same. But when a person attempted larger number of times, responses are considerably differing. Considering this pattern of standard deviation, the input can be safely said to be a suitable sample for investigation.

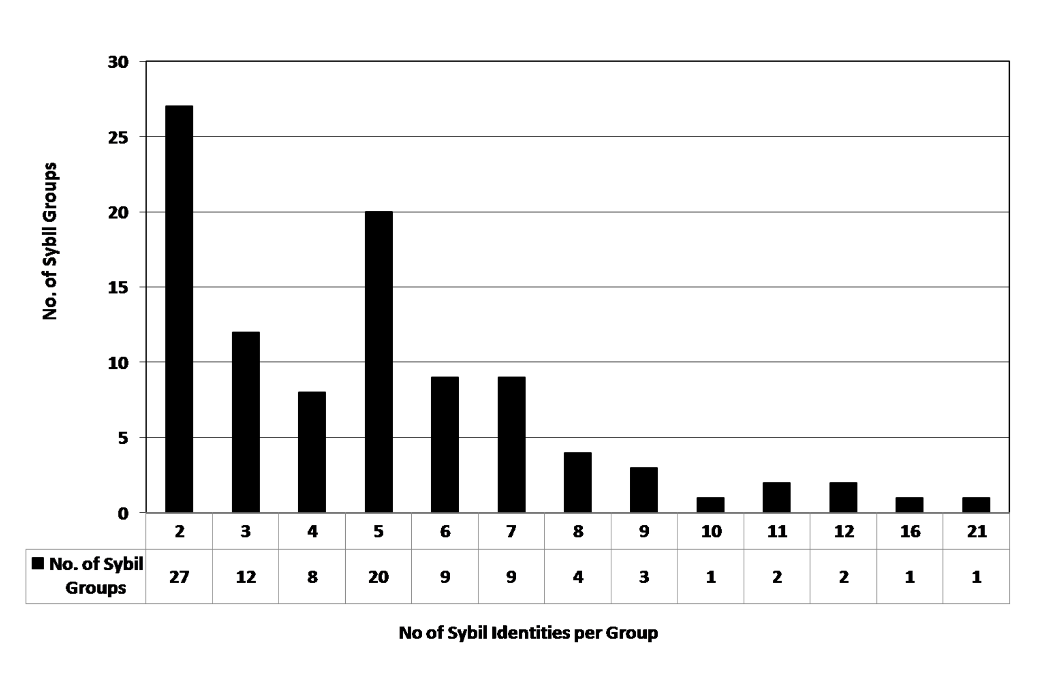
5.4.6 Results Analysis

The results are analysed with the objective of how good are the psychometric tests to detect Sybil groups in an overlay network. There are two types of data to analyse: rankings given to Luscher 8 colors and answers chosen for MBTI questionnaires. The similarity metrics for ranking data are Kendall's coefficient τ [Kendall & Smith 1939] and Spearmans



	1	2	3	4	5	6	7	8	9	10	11	12	16	21
■ % of Participants	46.49	14.59	6.49	4.32	10.81	4.86	4.86	2.16	1.62	0.54	1.08	1.08	0.54	0.54

Figure 5.15: Survey participation statistics - one attempt means a participant has answered the questionnaire once



	2	3	4	5	6	7	8	9	10	11	12	16	21
■ No. of Sybil Groups	27	12	8	20	9	9	4	3	1	2	2	1	1

Figure 5.16: Sybil Groups in the mapped network - one attempt by participant is mapped to one identity in peer-to-peer network

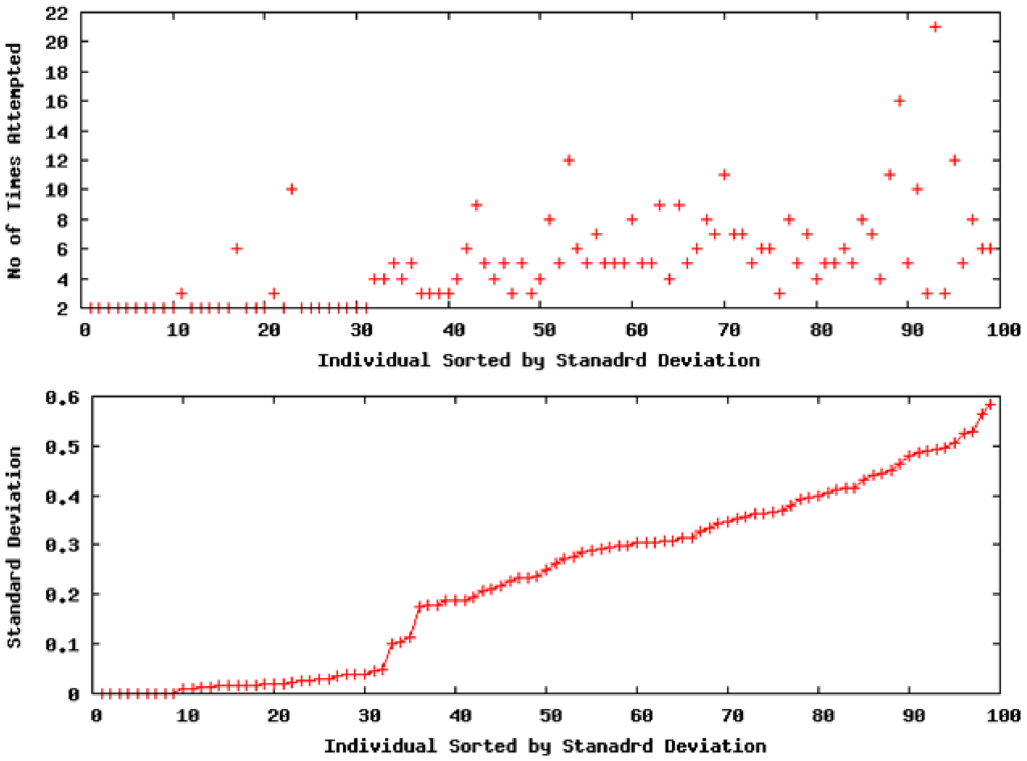


Figure 5.17: Standard deviation of responses per individual - standard deviation is computed for Spearman rank correlation coefficient

Correlation Coefficient ρ [Spearman 1987]. DBSCAN algorithm is run on the ranking data collected in each questionnaire for ϵ taking values in [0,1] each time incremented by 0.01 and *minPoints* taking values from 1 to 5. For each run, cluster goodness indices Rand statistic, Jaccard Coefficient and Fowlkes and Mallows index are measured. Figures 5.18, 5.19, and 5.20 show Rand Statistic, Jaccard coefficient and Fowlkes and Mallows index drawn for Kendall's τ. It can be observed that the graphs reach the highest values at a particular combination. Similar graphs for other similarity measures are presented in Appendix B.3.

Tables 5.8, 5.9 and 5.10 show the values of ϵ and *minPoints* at the best recorded values of Rand, Jaccard and Fowlkes and Mallows indices respectively. Similarly for the MBTI, the similarity indexes are Cosine Similarity Metric and Pearson Correlation Coefficient. The index's values and the best ϵ and *minPoints* are also shown in table 5.8, 5.9 and 5.10. Rest of the analysis is carried out only at the values of ϵ and *minPoints* listed in tables 5.8, 5.9 and 5.10.

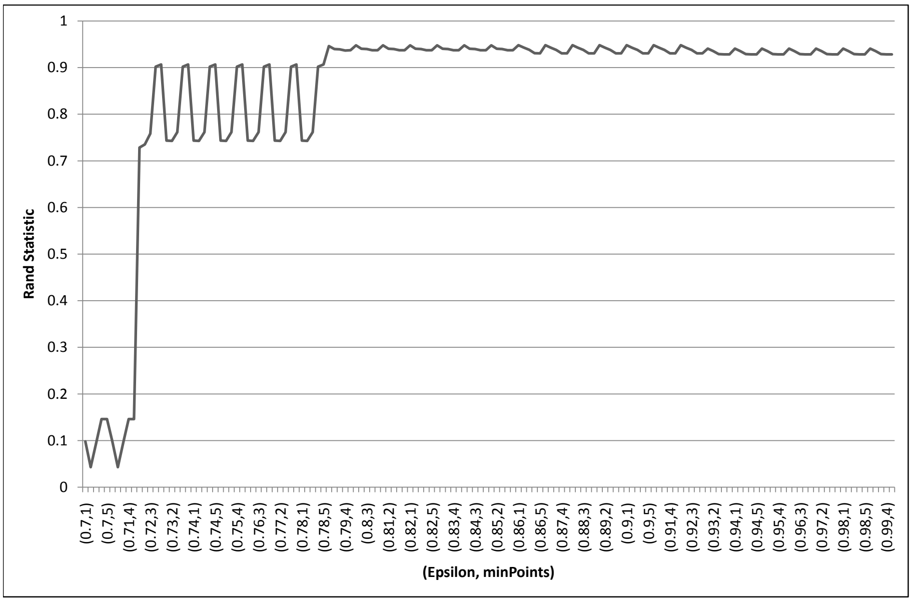


Figure 5.18: Values of Rand Statistic for Kendall's τ - Rand statistic is calculated for various combinations of ϵ and minPoints

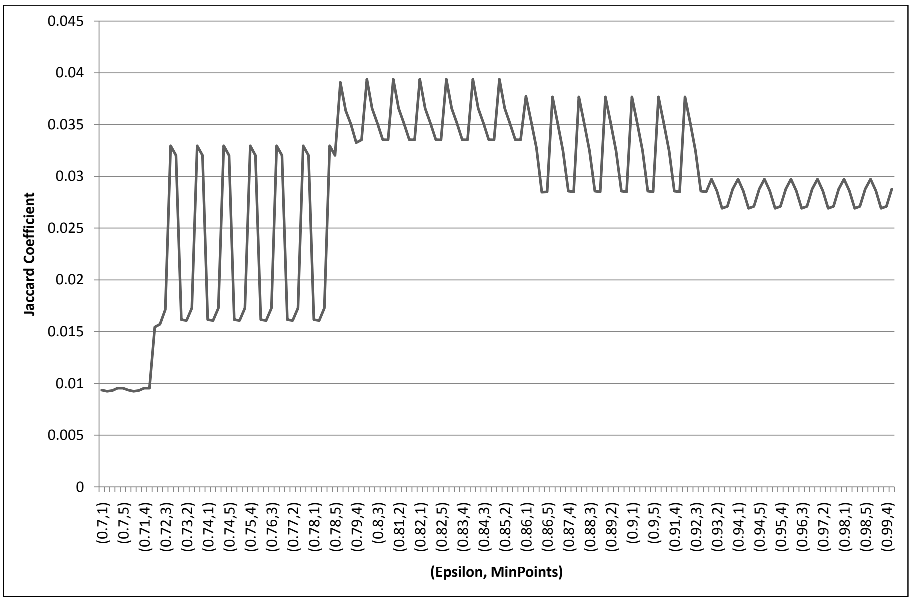


Figure 5.19: Values of Jaccard Statistic for Kendall's τ - Jaccard statistic is calculated for various combinations of ϵ and minPoints

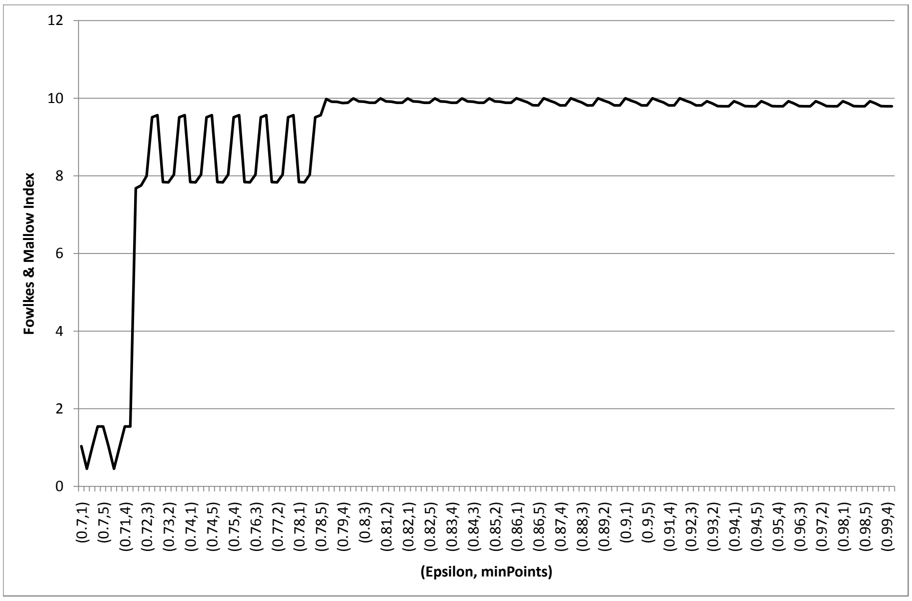


Figure 5.20: Values of Fowlkes & Mallow's index for Kendall's τ - Fowlkes & Mallow's index is calculated for various combinations of ϵ and minPoints

Table 5.8: Values of Epsilon and *minPoints* for DBSCAN algorithm at the best cluster formation stage i.e. at the maximum Rand Statistic value

Test	**Neighbour Distance Metric**	**Rand Statistic**		
		Epsilon	**MinPoints**	**Value**
Luscher Short Color Test	Kendall's Tau (K)	0.86	1	0.948
Luscher Short Color Test	Spearman's Rho (S)	0.94-0.95	1	0.9682
MBTI	Cosine Similarity Metric (C)	0.95-0.99	1	0.95
MBTI	Pearson Correlation Coeff (P)	0.43-0.44	1	0.829

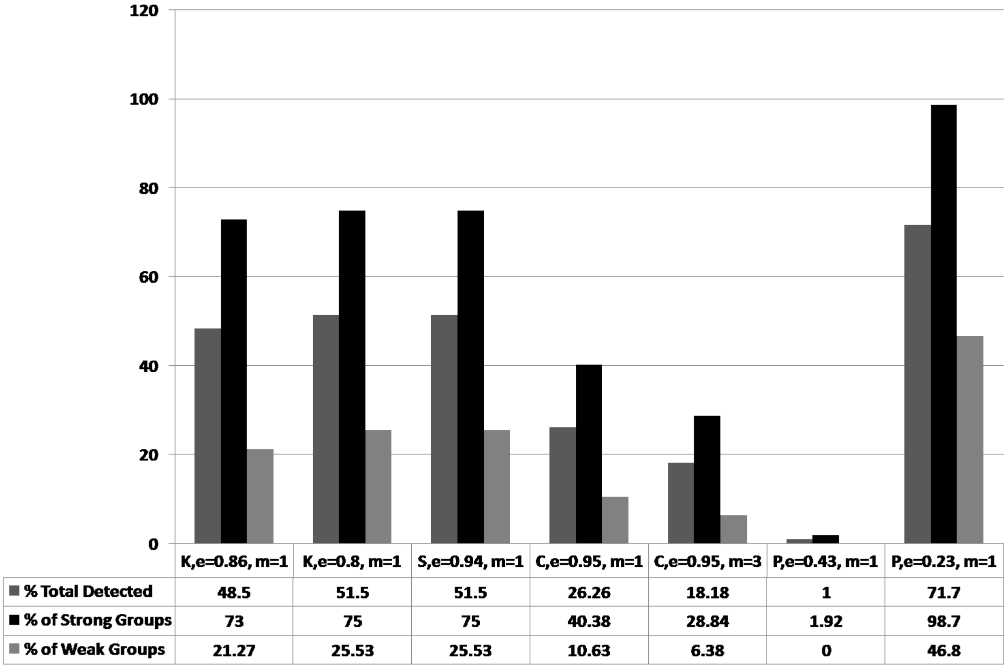
Table 5.9: Values of Epsilon and *minPoints* for DBSCAN algorithm at the best cluster formation stage i.e. at the maximum Jaccard Coefficient

Test	**Neighbour Distance Metric**	**Jaccard Coefficient**		
		Epsilon	**MinPoints**	**Value**
Luscher Short Color Test	Kendall's Tau (K)	0.80-0.85	1	0.0394
Luscher Short Color Test	Spearman's Rho (S)	0.94-0.95	1	0.0578
MBTI	Cosine Similarity Metric (C)	0.95-0.99	3	0.0121
MBTI	Pearson Correlation Coeff (P)	0.23-0.29	1-5	0.0133

Table 5.10: Values of Epsilon and *minPoints* for DBSCAN algorithm at the best cluster formation stage i.e. at the maximum Fowlkes & Mallows Index

Test	**Neighbour Distance Metric**	**Fowlkes & Mallows Index**		
		Epsilon	**MinPoints**	**Value**
Luscher Short Color Test	Kendall's Tau (K)	0.86	1	9.996
Luscher Short Color Test	Spearman's Rho (S)	0.94-0.95	1	10.211
MBTI	Cosine Similarity Metric (C)	0.95-0.99	1	10.019
MBTI	Pearson Correlation Coeff (P)	0.43-0.44	1	8.751

5.4.6.1 Effectiveness in Detecting Sybil Groups



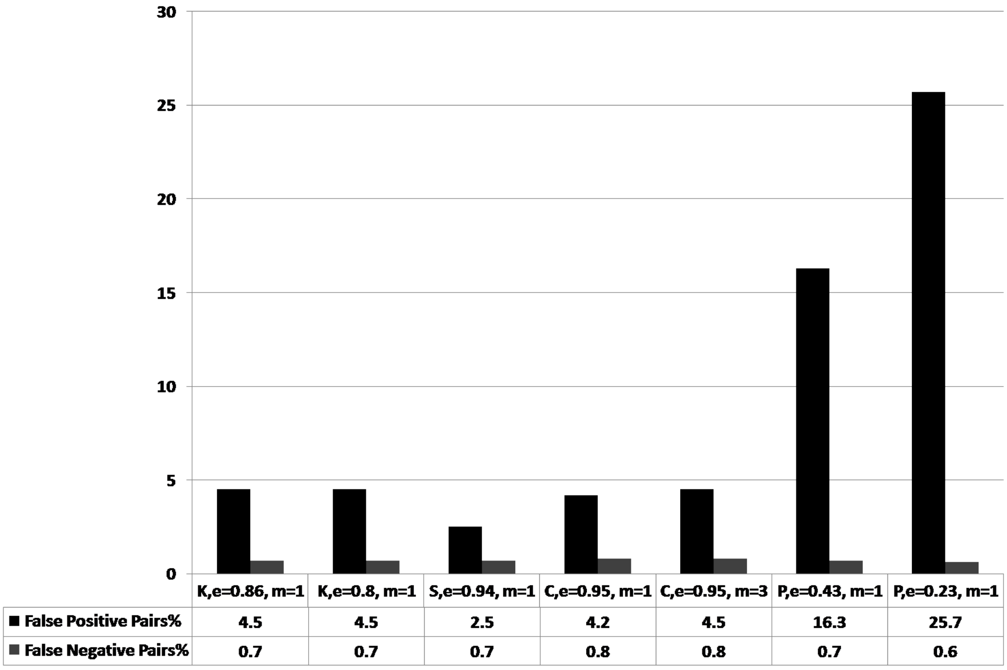
	K,e=0.86, m=1	K,e=0.8, m=1	S,e=0.94, m=1	C,e=0.95, m=1	C,e=0.95, m=3	P,e=0.43, m=1	P,e=0.23, m=1
■ % Total Detected	48.5	51.5	51.5	26.26	18.18	1	71.7
■ % of Strong Groups	73	75	75	40.38	28.84	1.92	98.7
■ % of Weak Groups	21.27	25.53	25.53	10.63	6.38	0	46.8

Figure 5.21: Percent of Sybil (Weak and Strong) groups detected when running clustering algorithm for different metrics - K, S, C and P refer to neighbour distance metrics. *e* means epsilon and *m* means minimum points

The results show that our method is able to detect 51.5% of the total Sybil groups. 75% of strong Sybil groups are detected. The percent of Sybil groups detected per each metric is shown in figure 5.21. Although cluster formation with Pearson metric discover 71.7% of the Sybil groups, it has got other limitations that all the Sybil groups are spread in just 4 clusters and the number of false positives is too high totalling to 25.7%. The false positives are shown in figure 5.22. That means that those four clusters are crowded with unnecessary identities. So Pearson correlation metric is not much useful in cluster formation here. It can be observed that it is difficult to detect weak Sybil groups compared to strong Sybil groups. This is depicted in figures 5.21. This is due to their small number of identities per group. One more thing to notice here is that more the number of identities in a Sybil group, more these tests can be consistent.

5.4.6.2 Luscher Color Test Vs MBTI Test

The results show that Lusher′s colour test gave better results compared to MBTI test. This is depicted in figure 5.21, and 5.23. This may be due to the fact that in every questionnaire



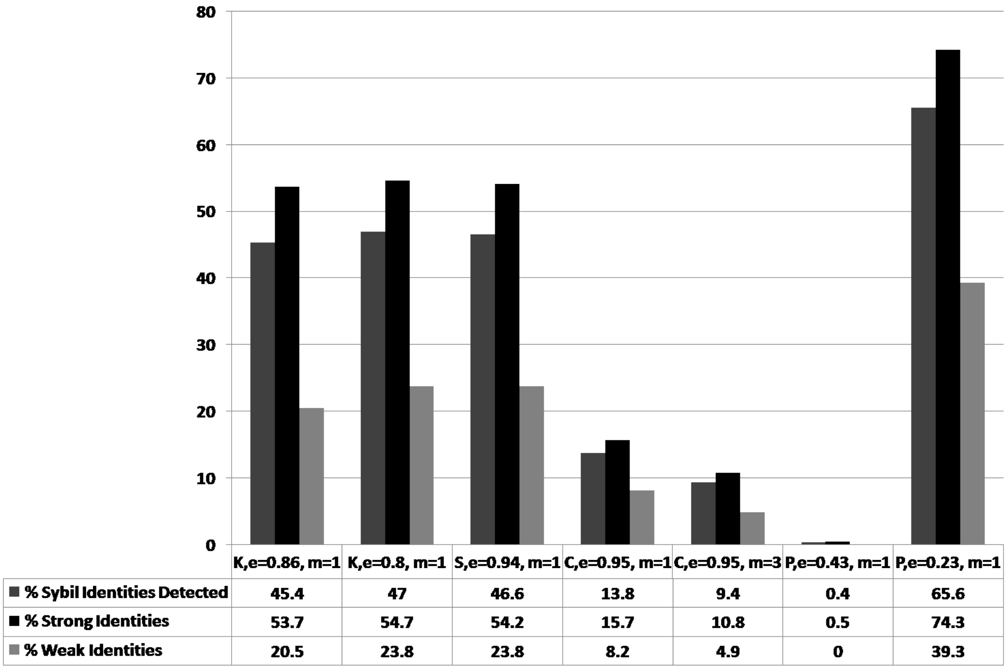
	K,e=0.86, m=1	K,e=0.8, m=1	S,e=0.94, m=1	C,e=0.95, m=1	C,e=0.95, m=3	P,e=0.43, m=1	P,e=0.23, m=1
■ False Positive Pairs%	4.5	4.5	2.5	4.2	4.5	16.3	25.7
■ False Negative Pairs%	0.7	0.7	0.7	0.8	0.8	0.7	0.6

Figure 5.22: Percent of false positives when clustering algorithm is run for different metrics - K, S, C and P refer to neighbour distance metrics. *e* means epsilon and *m* means minimum points

the colour set remained the same but the MBTI questions were not. The MBTI questions were selected from each functional pair randomly. The rankings given to colours were mostly consistent but the options chosen for the questions were not consistent.

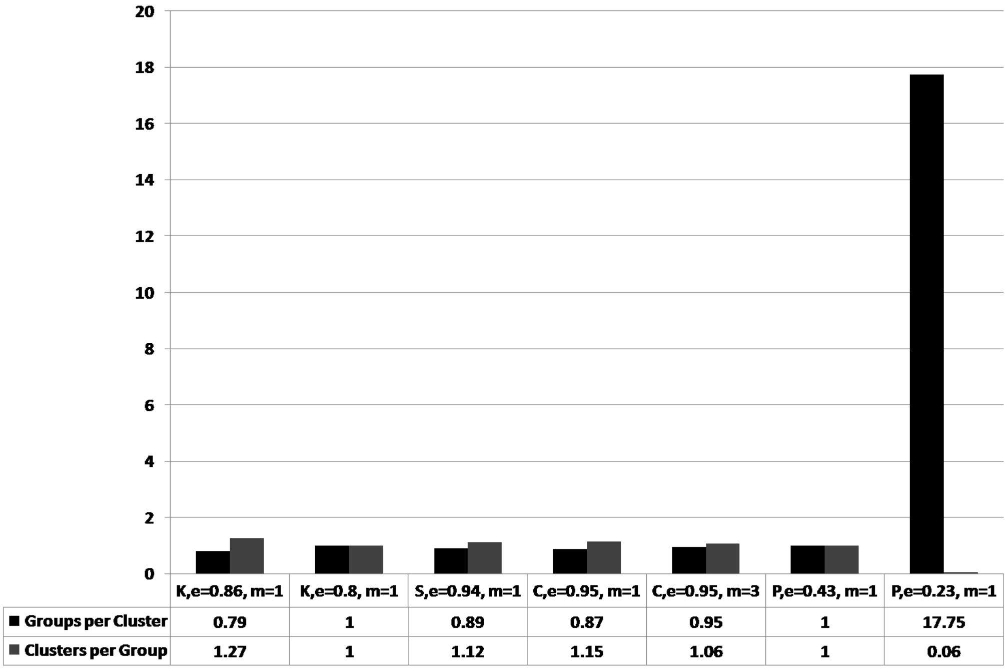
5.4.6.3 Quality of Clusters (Sybil Groups) Detected

The quality of the Sybil groups detected can be observed from two perspectives. One is that if the ratio of clusters to groups is near to 1, it means that the whole cluster is dedicated to one Sybil group. This eases the further process of issuing computational puzzles simultaneously to all identities in one Sybil group. As shown in figure 5.24, most of the metrics except the Pearson metric, the ratio comes near to 1. Second perspective is to see how much of a Sybil group is discovered. If most of the identities of a Sybil group are discovered that means all the identities will be issued computational puzzles putting real constraint on the resources of the Sybil entity. Figure 5.25 shows that for Luscher's test, roughly 65% of the group is detected in strong group category.



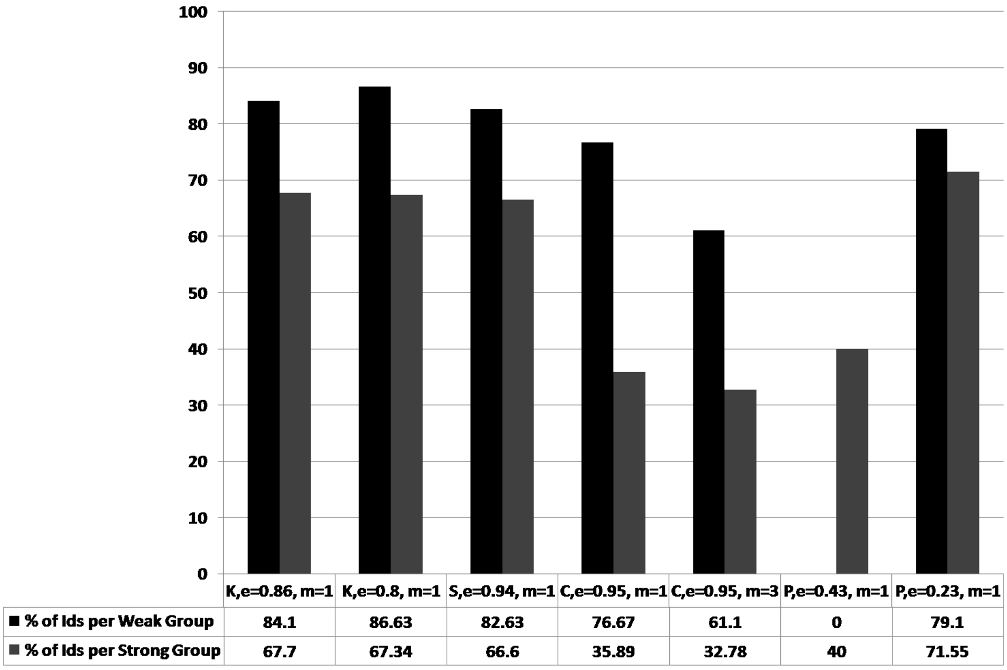
	K,e=0.86, m=1	K,e=0.8, m=1	S,e=0.94, m=1	C,e=0.95, m=1	C,e=0.95, m=3	P,e=0.43, m=1	P,e=0.23, m=1
■ % Sybil Identities Detected	45.4	47	46.6	13.8	9.4	0.4	65.6
■ % Strong Identities	53.7	54.7	54.2	15.7	10.8	0.5	74.3
■ % Weak Identities	20.5	23.8	23.8	8.2	4.9	0	39.3

Figure 5.23: Percent of Sybil identities (weak and strong) when clustering algorithm is run for different metrics - K, S, C and P refer to neighbour distance metrics. *e* means epsilon and *m* means minimum points



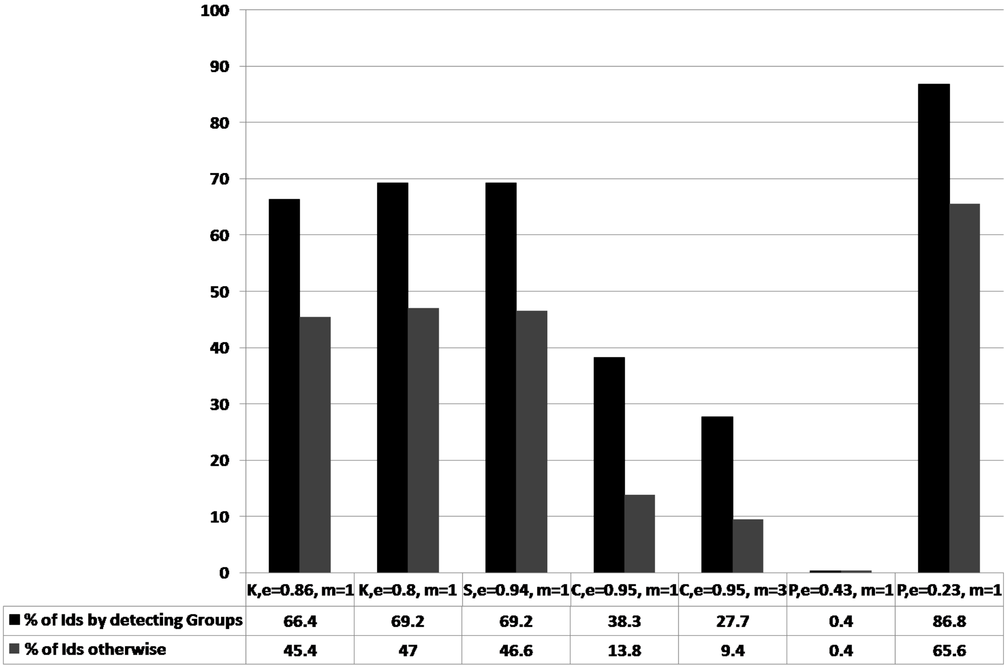
	K,e=0.86, m=1	K,e=0.8, m=1	S,e=0.94, m=1	C,e=0.95, m=1	C,e=0.95, m=3	P,e=0.43, m=1	P,e=0.23, m=1
■ Groups per Cluster	0.79	1	0.89	0.87	0.95	1	17.75
■ Clusters per Group	1.27	1	1.12	1.15	1.06	1	0.06

Figure 5.24: Spread of Sybil groups across clusters - K, S, C and P refer to neighbour distance metrics. *e* means epsilon and *m* means minimum points



	K,e=0.86, m=1	K,e=0.8, m=1	S,e=0.94, m=1	C,e=0.95, m=1	C,e=0.95, m=3	P,e=0.43, m=1	P,e=0.23, m=1
■ % of Ids per Weak Group	84.1	86.63	82.63	76.67	61.1	0	79.1
■ % of Ids per Strong Group	67.7	67.34	66.6	35.89	32.78	40	71.55

Figure 5.25: Percent of Identities detected in the detected Sybil groups - K, S, C and P refer to neighbour distance metrics. *e* means epsilon and *m* means minimum points



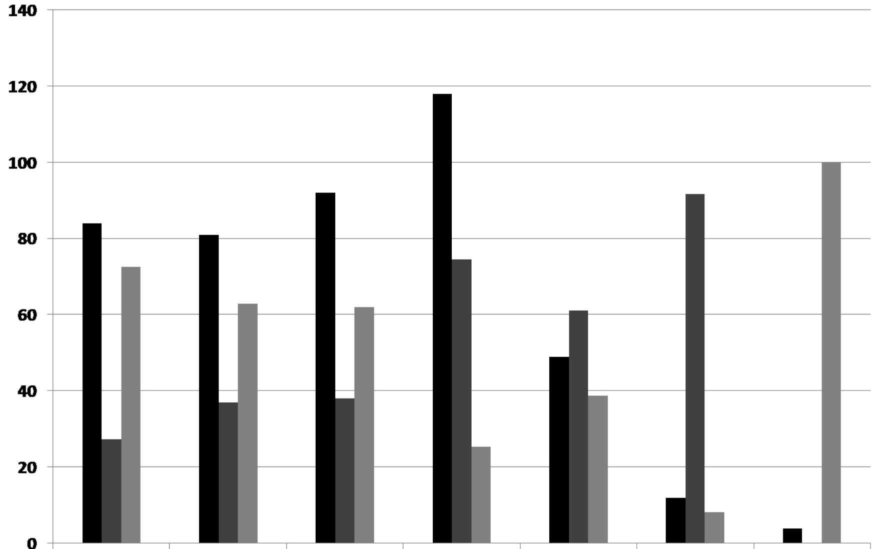
	K,e=0.86, m=1	K,e=0.8, m=1	S,e=0.94, m=1	C,e=0.95, m=1	C,e=0.95, m=3	P,e=0.43, m=1	P,e=0.23, m=1
■ % of Ids by detecting Groups	66.4	69.2	69.2	38.3	27.7	0.4	86.8
■ % of Ids otherwise	45.4	47	46.6	13.8	9.4	0.4	65.6

Figure 5.26: Depiction of the gain in % of Sybil identities detected through detecting Sybil groups - K, S, C and P refer to neighbour distance metrics. *e* means epsilon and *m* means minimum points

5.4.6.4 Detecting Sybil Groups Vs Detecting Sybil Identities

It can be observed from figures 5.21 and 5.23, detecting Sybil groups has advantages compared to detecting Sybil identities. For example, using Kendall's eps=0.80, the number of Sybil groups discovered are 51 and the number of Sybil identities in all these groups are 340. For the same, the number of Sybil identities discovered is only 231 which is 47% of the total. If we go by discovering Sybil groups, we can end up removing 69% of the Sybil identities from the network. These comparisons are depicted in figure 5.26.

5.4.6.5 Barren Clusters



	K,e=0.86, m=1	K,e=0.8, m=1	S,e=0.94, m=1	C,e=0.95, m=1	C,e=0.95, m=3	P,e=0.43, m=1	P,e=0.23, m=1
■ No of Clusters	84	81	92	118	49	12	4
■ % of Barren Clusters	27.4	37	38	74.6	61.2	91.7	0
■ % of Fruitful Clusters	72.6	63	62	25.4	38.8	8.3	100

Figure 5.27: Cluster statistics for different metrics - K, S, C and P refer to neighbour distance metrics. *e* means epsilon and *m* means minimum points

Also we can observe from figure 5.27 that the number of clusters discovered doesn't hold as much significance as the number of Sybil groups discovered, because a good percent of clusters are barren. From the Personality server system's point of view, it doesn't have the knowledge of Sybil groups. It has the information of clusters only. Therefore the relationship between number of clusters and number of Sybil groups holds significance. As depicted in figure 5.24, if the ratio comes to 1, that is the best. Then the system can understand that each cluster is a Sybil group and then act accordingly.

5.4.7 Conclusion

In this work we have presented a novel approach to detecting Sybil groups using psychometric tests. A survey is conducted amongst students and faculty in the campus. The experimental results have shown that 75% of the strong Sybil groups were detected. This shows that it is feasible to use psychometric tests to detect Sybil groups. The effectiveness of the test is also proved by detecting on average 67% of Sybil identities in Sybil group. Mechanisms are developed to safe-guard from Sybil strategies to subvert the solution.

5.5 Conclusion

In this chapter, two approaches are proposed for detecting Sybils: (i) Storage constraint based challenge-response approach and (ii) detecting Sybil groups based on psychometric tests approach. The first one is a fully distributed approach and the the one needs a central component. In storage-constraint approach, Sybil's attempt to drop, corrupt replicas is detected and actions are taken to keep the replicas safe. In psychometric tests approach, Sybil entity or group is detected and challenges are issued.

The following outlines the contexts in which they can be suitable.

- **Context of application for Storage constraint based challenge-response approach**: This approach is fully dependent on file replication. A network where large number of files are shared and replicated evenly on neighbours will be suitable.

- **Context of application for Psychometric tests approach**: Any network which can expect users to fill in the questionnaires regularly.

Chapter 6

Conclusion

6.1 Conclusions

Two objectives addressed in this thesis work are

1. To propose efficient algorithms for object search in Peer-to-Peer Overlays
2. To propose novel algorithms to safeguard against Sybil attacks in Peer-to-Peer Overlays

In Chapter 2, these problems are explained and the solutions in the literature were discussed. Chapter 3 and chapter 4 addressed first objective and chapter 5 addressed the second objective. The following are the conclusions.

1. In Chapter 2, factors that affect search in unstructured overlays are identified. They are topology adaptation, object placement and routing. In routing, further factors are neighbour selection, TTL and indexing.

2. In Chapter 3, a new approach for neighbour selection is proposed. It is identified that the previous approaches focussed on historical performance as the basis of neighbour selection. Historical performance alone can't select neighbours because each query is looking for a different content. Therefore we proposed a approach which takes nature of the query into account. Two parameters, namely what type of content query is looking for and how popular is it are considered to guide the query. Since these two parameters are very subjective, Fuzzy Sets are used to represent

them. This approach is proved to be better than existing approaches. Seeing its limitations and advantages of maximum-node-degree approach, they are combined to derive a hybrid metric. It is showed that this hybrid metric gave the best results.

3. In Chapter 3, an indexing technique is proposed to improve search in unstructured networks. This indexing doesn't need extra messages. The propagation of the indexes and the updates on them is done through query traffic. Space efficient data structures are designed using Bloomfilters. This indexing technique is named as 'Floating Indexes'. Analysis on different carrier methods like flooding, random walk, fuzzy walkers is carried out. Also the two ways of distributing indexes namely breadth-wise and depth-wise are analysed. In conclusion, breadth-wise spreading through random walkers is the better efficient way of indexing. Also the technique is compared to attenuated Bloomfilters indexing technique and found to be more efficient.

4. User satisfaction in search can't be fully addressed unless user requirements are taken into consideration. Two users may be searching for the same file, one with the intention of downloading it and the other with the intention of playing it through streaming. If the search is solely based on keywords, then these requirements are missed out and the results may disappoint the user. In Chapter 4, a QoS-search model is presented with performance evaluation. Network parameters namely bandwidth, latency and overlay parameter object locality are mapped to user expectations namely file downloads, streaming and document retrieval. Search is guided by the QoS constraints set by the user. Cost metric is designed to help the user select the result that best matches his expectations. This model is evaluated and it is proved to be providing results meeting user requirements. Our search algorithm is compared with SAMCRA, a centralised multi-constraint QoS algorithm. It is found that our algorithm has got results almost close to that of SAMCRA although our algorithm is distributed in all aspects.

5. In Chapter 2, a clear analysis of Sybil problem is presented. The solutions proposed in literature are classified and discussed. The solutions are categorised into challenge-response imposing constraints on resources, binding the identity to phys-

ical characteristics, central authority certification, characteristics of social networks based on trusted connections, based on Sybil behavioural characteristics and incentives.

6. Challenge-response model relies on the principle that since Sybil means multiple identities originating from a single physical entity, if each identity is given a task demanding the entire resources of one single physical entity, Sybils will fail to answer the challenge. In Chapter 5, storage constraint challenge-response approach is proposed. The storage constraint has an advantage compared to other constraints such as computational power and memory because in other constraints all Sybil identities need to be issued challenges simultaneously. In the case of storage constraint, the challenges can be spread over a period of time. A model centred on storage constraint is proposed. The relationship between number of replicas, proportion of Sybils are analysed.

7. An approach first of its kind based on Psychometric tests is proposed in chapter 5 as a solution to Sybil problem. It is based on the principle that Sybil identities originate from an individual person. Therefore there should be some common psychological characteristics among the Sybil identities. Through experimental means, it is proven to be very effective. It has the capability to detect Sybil group as a whole instead of working on detecting Sybil identities.

6.2 Summary of Contributions

The following are the contributions of the research carried out as a part of this thesis work.

1. Developed a content-oriented neighbour selection metric using Fuzzy Sets.

2. Developed a hybrid metric combining Fuzzy Set approach and maximum-node-degree approach.

3. Developed an indexing technique 'Floating Indexes', very apt for high query traffic in peer-to-peer networks, that doesn't need any extra messages.

4. Analysed in detail the ways of transporting indexes through queries.

5. Developed a user-centric QoS search model for peer-to-peer file sharing networks.

6. Developed a routing mechanism to find paths that satisfy multiple constraints in peer-to-peer file sharing systems.

7. Developed a challenge-response solution to detect Sybil identities that doesn't need simultaneous challenges to be issued.

8. Developed a first of its kind, Psychometric Tests based solution for detecting Sybil groups.

6.3 Future Research

The following are the areas that need further research.

1. A mathematical approach to model the query traffic in peer-to-peer overlays is required to provide guarantees on performance using indexing techniques.

2. Finding paths with QoS with local knowledge alone doesn't lead to optimal paths. Techniques such as landmarks based routing etc can give better results.

3. Storage-constraint based challenge-response model and Psychometric tests based model need to be deployed into a real peer-to-peer network and measure their performance.

Appendix A

Simulator Source

The simulator model is explained in section 3.2.3.1. It has two main classes namely, Simulator Main and Node.

A.1 Simulator Main

Here is the source code of the Main class that loads the topology and setsup the network.

Listing A.1: Simulator Main

```
package floatingIndexes;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;
import java.util.Random;
import java.util.StringTokenizer;
import javax.swing.Timer;
import Common.PoissonDistribution;
import Jama.*;

public class Main {
  static int lastHighestNodeNo = 0;
  static long startTime = System.currentTimeMillis();
  static int msgid = 0;
  static int qindex = 0;
  static int exitnode_count = 0;
  static float kwwtsum = 0;
  static String kw[] = {         ,        ,       ,       ,       ,       ,
      ,... };

  static String filetype[] = {       ,      ,      ,      ,       ,      ...
      };

  /**
   * A-audio D-Document G-Program R-Archive P-Picture V-Video
   */
  static String filecat[] = {    ,    ,    ,    ,    ,    ,    ,    ,    ...
      };

  /**
   * P - Populart, N-Normal, R-rare
   */
  static String filepopularity[] = {    ,    ,    ,    ,    ,    ,    ,
      ... };

  static String downloadtype[] = {    ,    ,    ,    ,    ,    ,    ,
      ...};

  static ArrayList<String> kwlist = new ArrayList<String>();
  static ArrayList<Integer> kwwt = new ArrayList<Integer>();
  static ArrayList<Integer> qperkw = new ArrayList<Integer>();
  static ArrayList<Node> al = new ArrayList<Node>();
  static Timer t = null;
  static Timer nodeRedist = null; // for node redistribution at regular
  // intervals of time
  static Timer objectRedist = null; // for object redistribution at regular
      intervals of time
  static int totalqueries = 0;
  /*
   * Weights
   */
  static Matrix wts = null;
  public static void main(String args[]) {
    System.out.println(         );
    String app = Constants.Application;
  if (app.equals(             ))
      wts = Weights.getWeights(new Matrix(Constants.pairwisematrix_d));
    if (app.equals(           ))
      wts = Weights.getWeights(new Matrix(Constants.pairwisematrix_s));
    if (app.equals(         ))
      wts = Weights.getWeights(new Matrix(Constants.pairwisematrix_v));
    if (app.equals(       ))
      wts = Weights.getWeights(new Matrix(Constants.pairwisematrix_e));
    if (wts.get(0, 0) == 0)
      System.exit(0);

/**
* Creating Node objects
*/

  for (int i = 0; i < Constants.MAX_NODES; i++) {
      Node n = new Node(i);
      al.add(n);
    }
/**
* Loading topology information
*/
    lastHighestNodeNo = al.size() - 1;
    BufferedReader br = null;
    try {
      br = new BufferedReader(new FileReader(                ));
    } catch (FileNotFoundException e) {
    }
    String line;
    try {
      while ((line = br.readLine()) != null) {
        int fnode = -1;
        int tnode = -1;
        int wt = 0;
        int wt1 = 0, wt2 = 0, wt3 = 0;
        StringTokenizer st = new StringTokenizer(line);
      if (st.countTokens() == 6) {
          fnode = Integer.parseInt(st.nextToken());
          tnode = Integer.parseInt(st.nextToken());
          wt = Integer.parseInt(st.nextToken());
          wt1 = Integer.parseInt(st.nextToken());
          wt2 = Integer.parseInt(st.nextToken());
          wt3 = Integer.parseInt(st.nextToken());

        }
        al.get(fnode).addNeighbour(al.get(tnode));
        al.get(fnode).setWeight(al.get(tnode), wt);
        al.get(fnode).setWeight2(al.get(tnode), wt2);
        al.get(fnode).setWeight3(al.get(tnode), wt3);
        al.get(tnode).addNeighbour(al.get(fnode));
        al.get(tnode).setWeight(al.get(fnode), wt);
        al.get(tnode).setWeight2(al.get(fnode), wt2);
        al.get(tnode).setWeight3(al.get(fnode), wt3);
      }
    } catch (IOException e) {
      System.out.println(e);
    }
/**
* Loading the objects
*/

    BufferedReader br0 = null;
    try {
      br0 = new BufferedReader(new FileReader(

                                + Constants.INPUT_SET +             ));
    } catch (FileNotFoundException e) {
      System.out.println(e);
    }
    String line0;
    try {
      while ((line0 = br0.readLine()) != null) {
        int i, k;
        String[] line_token1 = line0.split(   );
        k = Integer.parseInt(line_token1[0]);
         kwlist.add(k, line_token1[1]);
        //al.get(i).addItem(new Item(new KeyWord(kwlist.get(k)), filecat[k],
            filepopularity[k]));
      }
    } catch (Exception e) {
      e.printStackTrace();
      System.out.println(e);
    }
/**
* Replicating the objects
*/

    BufferedReader br1 = null;
    try {
      br1 = new BufferedReader(new FileReader(

                                + Constants.INPUT_SET +                   ));
    } catch (FileNotFoundException e) {
```

```
   System.out.println(e);
  }
  String line1;
  try {
   while ((line1 = br1.readLine()) != null) {
    int i, k;
    String[] line_token1 = line1.split(   );
    k = Integer.parseInt(line_token1[0]);
    i = Integer.parseInt(line_token1[1]);
    al.get(i).addItem(new Item(new KeyWord(kwlist.get(k)),   ,   ));
   }
  } catch (Exception e) {
   e.printStackTrace();
   System.out.println(e);
  }

/**
* Loading queries from file
*/
  final int queryorder[][] = new int[10100][2];
  BufferedReader br2 = null;
  try {
   String line2;
   br2 = new BufferedReader(new FileReader(

                + Constants.INPUT_SET +
       ));
   int j = 0;
   while ((line2 = br2.readLine()) != null) {
    int i, k;
    String[] line_token2 = line2.split(   );
    k = Integer.parseInt(line_token2[0]);
    i = Integer.parseInt(line_token2[1]);
    queryorder[j][0] = k;
    queryorder[j][1] = i;
    j++;
   }
  }
  catch (Exception e) {
   System.out.println(e);
  }

/**
* Starting node threads
*/
  for (int i = 0; i < al.size(); i++) {
   Thread t = new Thread(al.get(i), i +   );
   al.get(i).th = t;
   t.start();
  }

/**
* Firing queries
*/

  t = new Timer(100, new ActionListener() {
   public void actionPerformed(ActionEvent ae) {
    if(qindex%500==0)System.out.println(qindex +          );
    Query m = new Query();
    m.msgid = msgid++;
    m.responseTime = 0;
```

```
      m.hopcount = Constants.MAX_HOPS;
      m.kw = new KeyWord(kwlist.get(queryorder[qindex][1]));
      m.msgType = 0;
      m.filetype = filetype[queryorder[qindex][1]];
      m.downloadtype = Constants.Application;
      m.cost = 0;
      m.inversepopularity = (float) (queryorder[qindex][1] + 1)
          / (float) (Constants.MAX_ITEMS + 1);

      m.fuzzfiletype=filecat[queryorder[qindex][1]];
      m.fuzzyfilepopularity=filepopularity[queryorder[qindex][1]];

      m.bw_weight = wts.get(0, 0);
      m.delay_weight = wts.get(1, 0);
      m.quckfind_weight = wts.get(2, 0);


       m.downloadtype=downloadtype[queryorder[qindex][1]];

      Node n = al.get(queryorder[qindex][0]);
      m.qoscost = m.bw_weight * n.getNormalizedBW(10)
          + m.delay_weight * n.getNormalizedAccDelay(100)
          + n.getNormalizedQFProb(.125) * m.quckfind_weight;
      m.delay_accumulated = 0;
      n.addMessage(n, m);

      if (qindex == 10000) {
        try {
          System.out.println(                );
          Thread.sleep(60000);
        } catch (Exception e) {
          System.out.println(e);
        }
        for (int i = 0; i < al.size(); i++) {
          al.get(i).isEndofAllQueries = true;
        }
        Log.isEndofAllQueries = true;
        LogPathEntries.isEndofAllQueries = true;
        t.stop();

      }
      qindex++;

      // }
    }
  });

  t.start();
  for (int i = 0; i < al.size(); i++) {
    try {
      al.get(i).th.join();
    } catch (Exception e) {
    }
  }


  System.out.println(             );

}

public static void computeKWWts() {
    double d = 0;
    double alpha = 0.83;
    for (int i = 0; i < kwlist.size(); i++)
      d = d + (1 / Math.pow(i + 1, alpha));
    System.out.println(d);
    // Assuming that the objects are ranked from 1 to 100
    for (int i = 0; i < kwlist.size(); i++) {
      kwwt.add(new Integer((int) Math
          .ceil(((1 / Math.pow(i + 1, alpha)) / d)
              * Constants.MAX_REPLICAS)));

    }
  }

}
```

A.2 Node

Here is the source code for the basic functionality of a node.

Listing A.2: Node

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Date;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Random;
import Common.BloomFilter;
import Common.Utils;
import Jama.Matrix;

public class Node implements Runnable {
  int nodeno;
  boolean visited=false;
  int wt = 0;
  public boolean isEndofAllQueries = false;
  Thread th = null;
  BloomFilter ngbf[][] = null;
  ArrayList<Node> ng = new ArrayList<Node>();
  ArrayList<Item> ic = new ArrayList<Item>();
  ArrayList<Integer> ngwt = new ArrayList<Integer>();
  ArrayList<Integer> bwwt = new ArrayList<Integer>();
  ArrayList<Integer> dlwt = new ArrayList<Integer>();
  ArrayList<MessageInfo> mc = new ArrayList<MessageInfo>(); // Message Cache
  ArrayList<Integer> wts = new ArrayList<Integer>();
/* Queue object for storing the incoming messages*/
  LinkedList<QueuedMessage> mq = new LinkedList<QueuedMessage>();// Message

  //fuzzy related
  /**0 A-audio
   * 1 D-Document
   * 2 G-Program
   * 3 R-Archive
   * 4 P-Picture
   * 5 V-Video
   */
  Matrix filestats=new Matrix (Constants.NO_OF_FILE_TYPES, Constants.
      NO_OF_POPULAR_CATEGORIES+1);
  Matrix fuzzyprob=new Matrix(Constants.NO_OF_FILE_TYPES, Constants.
      NO_OF_POPULAR_CATEGORIES+1);
  Matrix ngstats=null;

/**
 * floatingIndexes
 */
  ArrayList<NodeIndexRecord> indexCache=new ArrayList<NodeIndexRecord>();

  float ahplist[] = { 0.1f, 0.1f, 0.1f, 0.7f };
  double bf_levelprob[] = null;
  float matlist[][];
  float b[];
  float tkw;
  int bw[];
  int pr[];
  int dl[];
  double qfp[];
  int mean;
  float sd;
  private Random generator = new Random();
  public Node(int nodeno) {
    this.nodeno = nodeno;
  }

  void setWeight(Node n, int wt) {
    if (ngwt.size() < ng.size()) {
      for (int i = 0; i < ng.size(); i++) {
        ngwt.add(20);
      }
    }
    int index = -1;
    for (int i = 0; i < ng.size(); i++)
      if (ng.get(i).nodeno == n.nodeno) {
        index = i;
        break;
      }
    if (index == -1)
      return;
    try {
      ngwt.add(index, new Integer(wt));
    } catch (ArrayIndexOutOfBoundsException e) {
      ngwt.add(index, new Integer(wt));
    }
  }
  void setWeightBW(Node n, int wt) {
    if (bwwt.size() < ng.size()) {
      for (int i = 0; i < ng.size(); i++) {
        bwwt.add(20);
      }
    }
    int index = -1;
    for (int i = 0; i < ng.size(); i++)
      if (ng.get(i).nodeno == n.nodeno) {
        index = i;
        break;
      }
    if (index == -1)
      return;
    try {
      bwwt.add(index, new Integer(wt));
    } catch (ArrayIndexOutOfBoundsException e) {
      bwwt.add(index, new Integer(wt));
    }
  }

  void setWeightDelay(Node n, int wt) {
    if (dlwt.size() < ng.size()) {
      for (int i = 0; i < ng.size(); i++) {
        dlwt.add(20);
      }
    }
    int index = -1;
    for (int i = 0; i < ng.size(); i++)
      if (ng.get(i).nodeno == n.nodeno) {
        index = i;
        break;
      }
    if (index == -1)
      return;
    try {
      dlwt.add(index, new Integer(wt));
    } catch (ArrayIndexOutOfBoundsException e) {
      dlwt.add(index, new Integer(wt));
    }
  }

  int getWeight(Node n) {
    int index = -1;
    for (int i = 0; i < ng.size(); i++)
      if (ng.get(i).nodeno == n.nodeno) {
        index = i;
        break;
      }
    if (index == -1)
      return 0;

    return ngwt.get(index).intValue();
  }

  public boolean addNeighbour(Node n) {
    // check for duplicates
    Iterator<Node> itr = ng.iterator();
    while (itr.hasNext()) {
      Node n1 = itr.next();
      if (n1.nodeno == n.nodeno)
        return false;
    }
    ng.add(n);
    return true;
  }

private int getNeighbourIndex(Node n){
  for(int i=0;i<ng.size();i++){
    if(ng.get(i).nodeno==n.nodeno) return i;
  }
  return -1;
}

  synchronized public void addMessage(Node n, Message m) {
    m.responseTime = m.responseTime + n.getWeight(this);
    synchronized (mq) {
      mq.addLast(new QueuedMessage(m, n));
    }
/**
* Logging the message stats
*/
          Log.writeNodeCoverage(m.msgid +     + m.hopcount +    +n.nodeno+
                + this.nodeno+    +m.msgType +     + m.hopstravelled +    + m
              .fuzzfiletype +     + m.fuzzyfilepopularity+     );
    if (m.msgType == 2) {
      Log.writeQH(m.msgid, ((QueryHit) m).hitnode.nodeno, m.pathlength,
m.inversepopularity, m.cost, m.bw, m.dl, m.fuzzfiletype,
m.fuzzyfilepopularity, (long) Math.floor((System
    .currentTimeMillis() - Main.startTime) / 1000),
          m.responseTime);
    }
  }


/**
* Starting point for the thread. It processes the queued messages.
*/

  public void run() {


    bw = new int[ng.size()];
    dl = new int[ng.size()];
    qfp = new double[ng.size()];
    int y;
    for (y = 0; y < ng.size(); y++) {
      bw[y] = (int) bwwt.get(y);
      dl[y] = (int) dlwt.get(y);
    }



    /**neighbour stats*/
    /**0-avg hops taken by qh, 1-avg responsetime taken by result, 2-greatest
         no of results, 3-greatest no of mesgs received
*/
    ngstats=new Matrix(ng.size(), 4);
    for(int i=0;i<ngstats.getRowDimension();i++)
      for(int j=0;j<ngstats.getColumnDimension();j++){
        ngstats.set(i, j, 0);
        if(j==0||j==1) ngstats.set(i, j, 10000);
      }

    long msgCounter=0;
    while (true) {
      while (mq.size() > 0) {

        try {
          for (int i = 0; i < mq.size(); i++) {
            Message m = mq.get(i).m;
            msgCounter++;
            if (m.msgType == 0) {
              receiveQuery(mq.get(i).sender, (Query) m);
              int ngindex=getNeighbourIndex(mq.get(i).sender);
              if(ngindex!=-1) ngstats.set(ngindex, 3, ngstats.get(ngindex, 3)
                  +1);
            }
            if (m.msgType == 1) {
              if (mq.get(i).sender.nodeno != this.nodeno)
                receiveQueryHit(mq.get(i).sender, (QueryHit) m);
              int ngindex=getNeighbourIndex(mq.get(i).sender);
              if(ngindex!=-1) ngstats.set(ngindex, 3, ngstats.get(ngindex, 3)
                  +1);
            }
            if (m.msgType == 2) {
               this.addItem(new Item(m.kw, m.fuzzfiletype, m.
                   fuzzyfilepopularity));
               int ngindex=getNeighbourIndex(mq.get(i).sender);
               if(ngindex!=-1) {
                 if(ngstats.get(ngindex, 0)==10000) ngstats.set(ngindex, 0, m
                     .pathlength);
                 else ngstats.set(ngindex, 0, (ngstats.get(ngindex, 0)+m.
                     pathlength)/2);
                 if(ngstats.get(ngindex, 1)==10000) ngstats.set(ngindex, 1, m
                     .responseTime);
                 else ngstats.set(ngindex, 1, (ngstats.get(ngindex, 1)+m.
                     responseTime)/2);
               ngstats.set(ngindex, 2, ngstats.get(ngindex, 2)+1);
               ngstats.set(ngindex, 3, ngstats.get(ngindex, 3)+1);
               }
            }
            synchronized (mq) {
              mq.remove(i);
            }
          }
        } catch (Exception e) {
          System.out.println(e);
          e.printStackTrace();
        }
      }

      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      if (isEndofAllQueries && mq.size() == 0) {
        System.out.println(        + nodeno +           );
        int local=0, nonlocal=0;
        for(int i=0;i<indexCache.size();i++)
          if(indexCache.get(i).n.nodeno!=nodeno)
            nonlocal++;
            else local++;
        Log.writeNodeCache(nodeno +     + indexCache.size() +     + local+
             + nonlocal +     );
        break;
      }
    }
    Main.exitnode_count++;
    System.out.println(                              + Main.exitnode_count);
  }

/**
* Procedure for processing a query
*/
```

```
private void receiveQuery(Node sn, Query m) {
  int ni = 0;
  if (m.hopcount == 0)
    return;
  for (int i = 0; i < mc.size(); i++) {
    if (mc.get(i).msgid == m.msgid){
        Log.writeDuplicateMsgs(nodeno +     + sn.nodeno +     + m.msgid +
               + m.hopstravelled +    + m.fuzzfiletype +     + m.
            fuzzyfilepopularity +     );
      return;
    }
  }
  mc.add(new MessageInfo(m, sn, new Date()));

  /**
   * floatingIndexes
   */
  updateIndexcacheFromQuery(m);
  refreshIndexCacheforLocalItems();
  addIndexCacheToQuery(m);

  for (int i = 0; i < ic.size(); i++) {
    if (ic.get(i).matches(m.kw))
      ni++;
  }

  ArrayList<Node> al=searchIndexCache(m.kw);

  if ((ni > 0||al.size()>0) && sn.nodeno != this.nodeno) {

    for (int i = 0; i < m.hopstravelled; i++) {
      //LogPathEntries.writeMisc(m.msgid + "," + this.nodeno + "," + i
    }

    QueryHit hm = new QueryHit();
    hm.msgid = m.msgid;
    hm.kw = m.kw;
    hm.hopcount = Constants.MAX_HOPS + 2;
    hm.msgType = 1;
    hm.hitnode = this;
    hm.nof = ni;
    hm.bw = 0;
    hm.dl = 0;
    hm.pathlength = 1;
    hm.responseTime = m.responseTime;
    hm.downloadtype = m.downloadtype;
    hm.inversepopularity = m.inversepopularity;
    hm.cost = m.cost;
    hm.filetype = m.filetype;
    hm.fuzzfiletype=m.fuzzfiletype;
    hm.fuzzyfilepopularity=m.fuzzyfilepopularity;
    if (sn.nodeno == this.nodeno) {
      System.out.println(                          + nodeno);
      hm.msgType = 2;
    }
    sn.addMessage(this, hm);
  } else {
    /*
     *
     * Finding Quick find probability
```

```
       */

      NeighbourCost arr[] = null;
      double N=0;
      if(nodeno==sn.nodeno){
        arr=new NeighbourCost[ng.size()];
        N=ng.size();}
      else{
        arr=new NeighbourCost[ng.size()-1];
        N=ng.size()-1;
      }

      int k=0;
      for (int i = 0; i < ng.size(); i++) {
        if ( ng.get(i).nodeno!=sn.nodeno){
(m.fuzzfiletype, m.fuzzyfilepopularity);

      if(nodeno==sn.nodeno) {
        N=9;
      }else N=0;
      //N=ng.size();
      for ( int i=0; i <= Math.min(N, arr.length-1); i++) {

        Query q = m.clone();
        q.hopcount--;
        q.cost = q.cost + arr[i].cost;
        q.bandwidth[q.hopstravelled] = bw[arr[i].ngindex];
        q.delay[q.hopstravelled] = dl[arr[i].ngindex];
        q.qfp[q.hopstravelled] = qfp[arr[i].ngindex];
        q.hopcost[q.hopstravelled] = arr[i].cost;
        q.nodenos[q.hopstravelled] = nodeno;

        q.delay_accumulated=q.delay_accumulated+dl[arr[i].ngindex];


        q.hopstravelled++;
        ng.get(arr[i].ngindex).addMessage(this, q);
      }

    }
  }
/**
* Procedure for processing a query hit
*/

  private void receiveQueryHit(Node sn, QueryHit m) {
    Node tobesentto = null;
    for (int i = 0; i < mc.size(); i++) {
      if (mc.get(i).msgid == m.msgid) {
        tobesentto = mc.get(i).n;
        break;
      }
    }
    if (tobesentto == null)
      return;
    else {
      int i, j = 0;
      for (i = 0; i < ng.size(); i++) {
        if (ng.get(i) == sn)
          j = i;
      }
      if ((m.bw != 0) && (m.bw < bw[j]))
        ;
      else
        m.bw = bw[j];
      m.dl = m.dl + dl[j];
      if (tobesentto.nodeno == this.nodeno) {
        m.msgType = 2;
        addMessage(this, m);
      } else {
        m.pathlength = m.pathlength + 1;
        tobesentto.addMessage(this, m);
      }
    }
  }


  public String toString() {
    return nodeno +   ;
  }


}

class MessageInfo {
  public MessageInfo(Query m, Node sn, Date date) {
    msgid = m.msgid;
    n = sn;
    t = date;
  }

  int msgid;

  Node n;

  Date t;
}
/**
* A class for encapsulating the incoming message
*/

class QueuedMessage implements Comparator {
  public QueuedMessage(Message m2, Node n) {
    m = m2;
    sender = n;
  }

  Message m;

  Node sender;

  public int compare(Object arg0, Object arg1) {
    QueuedMessage a = (QueuedMessage) arg0;
    QueuedMessage b = (QueuedMessage) arg1;
    if (a.m.msgid > b.m.msgid)
      return 1;
    else if (a.m.msgid == b.m.msgid) {
      if (a.m.cost > b.m.cost)
        return 1;
      else if (a.m.cost < b.m.cost)
        return -1;
      else
```

```
        return 0;
      } else
        return -1;
    }
  }

  /**
  * A class for storing the cost for each neighbour for a particular query
  */
  class NeighbourCost implements Comparator {
    double cost;

    int ngindex;

    NeighbourCost(double cost, int ngindex) {
      this.cost = cost;
      this.ngindex = ngindex;
    }

    public int compare(Object arg0, Object arg1) {
      NeighbourCost a = (NeighbourCost) arg0;
      NeighbourCost b = (NeighbourCost) arg1;

      if (a.cost > b.cost)
        return -1; //1 ascedning //-1 descending
      else if (a.cost < b.cost)
        return 1; //-1 asc 1-desc
      else
        return 0;
    }

  }
```

A.3 Utility Functions

The following functions are coded to assistin fuzzy based and QoS based operations.

Listing A.3: Fuzzy and QoS reated functions

```
  /**
   * Fuzzy utility Functions
   */

    public double getFilePercent(String filetype, String filepopularity){
      int total=0,subtotal=0;
      for(int k=0; k<ic.size();k++)
        if(ic.get(k).fileCat.equals(filetype)){
          total++;
        if(ic.get(k).popularity.equals(filepopularity)) subtotal++;
        }
      if (total==0) return 0;
      return (subtotal/total)*100;
      }

    public double getFuzzyProb(String filetype, String filepopularity){
      int findex=-1, pindex=-1;
      for(int i=0;i<Constants.NO_OF_FILE_TYPES;i++)
      if(Constants.FILE_TYPES[i].equals(filetype)) findex=i;
    for(int i=0;i<Constants.NO_OF_POPULAR_CATEGORIES+1;i++)
      if(Constants.POPULAR_CATEGORIES[i].equals(filepopularity)) pindex=i;
    return fuzzyprob.get(findex, pindex);
    }

class FuzzyConfidenceValues{
  double fpl;
  double fpm;
  double fph;

  public void print(){
    System.out.println(            + fpl +           +fpm +           + fph);
  }

  public void print(double val){
    System.out.println(            +val +          + fpl +           +fpm +
                 + fph);
  }
    }

  public  FuzzyConfidenceValues getConfidenceValues(double tp){
    double fptl=0,fptm=0,fpth=0;

    //low confidence values
    if(tp>=0 && tp <=20) fptl=1;
    if(tp>=20 && tp <=80) fptl= (80-tp)/60;
    if(tp>80) fptl=0;

    //medium confidence values
    if(tp<20) fptm=0;
    if(tp>=20 && tp <=50) fptm=(tp-20)/30;
    if(tp>=50 && tp <=80) fptm= (80-tp)/(30);
    if(tp>80) fptm=0;

    //high confidence values
    if(tp<20) fpth=0;
    if(tp>=20 && tp <=80) fpth=(tp-20)/60;
    if(tp>=80 && tp <=100) fpth=1;
    if(tp>100) fpth=0;
    FuzzyConfidenceValues fcv=new FuzzyConfidenceValues();
    fcv.fph=fpth;
    fcv.fpl=fptl;
    fcv.fpm=fptm;
  return fcv;
  }

private void computeFuzzyProbabilities(){
  for(int i=0; i<filestats.getRowDimension();i++)
    for(int j=0;j<filestats.getColumnDimension();j++){
      double total=0;
      for(int k=0; k<ic.size();k++){
        if (j==0) if(ic.get(k).fileCat.equals(Constants.FILE_TYPES[i])) total
            ++;
        if(j>0) if(ic.get(k).fileCat.equals(Constants.FILE_TYPES[i]) && ic.
            get(k).popularity.equals(Constants.POPULAR_CATEGORIES[j])) total
            ++;
      }
      filestats.set(i, j, total);
    }
  for(int i=0; i<fuzzyprob.getRowDimension();i++)
    for(int j=0;j<fuzzyprob.getColumnDimension();j++)
      fuzzyprob.set(i, j, 0);
  for(int i=0; i<filestats.getRowDimension();i++){
    for(int j=1;j<filestats.getColumnDimension();j++){

      if(i>=0 && j>=0){
      double tp=ic.size();//(filestats.get(i,0)/ic.size())*100;
      FuzzyConfidenceValues tpfcv=getConfidenceValues(tp);
      double popcatp;
      if (filestats.get(i,0)==0)
        popcatp=0;
      else popcatp=(filestats.get(i,j)/filestats.get(i,0))*100;
      FuzzyConfidenceValues popfcv=getConfidenceValues(popcatp);
      double max=0;
      for(int k=0;k<ng.size();k++){
        //double x=ng.get(k).getFilePercent(Constants.FILE_TYPES[i],Constants
            .POPULAR_CATEGORIES[j]);
        double x=100 * ng.get(k).getFuzzyProb(Constants.FILE_TYPES[i],
            Constants.POPULAR_CATEGORIES[j]);
        if(x>max)max=x;
      }
      FuzzyConfidenceValues ngfcv=getConfidenceValues(max);
    Matrix ngl=new Matrix(3,3);
      ngl.set(0, 0, Math.min(Math.min(tpfcv.fpl,popfcv.fpl), ngfcv.fpl));
      ngl.set(0, 1, Math.min(Math.min(tpfcv.fpl,popfcv.fpm), ngfcv.fpl));
      ngl.set(0, 2, Math.min(Math.min(tpfcv.fpl,popfcv.fph), ngfcv.fpl));

      ngl.set(1, 0, Math.min(Math.min(tpfcv.fpm,popfcv.fpl), ngfcv.fpl));
      ngl.set(1, 1, Math.min(Math.min(tpfcv.fpm,popfcv.fpm), ngfcv.fpl));
      ngl.set(1, 2, Math.min(Math.min(tpfcv.fpm,popfcv.fph), ngfcv.fpl));

      ngl.set(2, 0, Math.min(Math.min(tpfcv.fph,popfcv.fpl), ngfcv.fpl));
      ngl.set(2, 1, Math.min(Math.min(tpfcv.fph,popfcv.fpm), ngfcv.fpl));
      ngl.set(2, 2, Math.min(Math.min(tpfcv.fph,popfcv.fph), ngfcv.fpl));

      Matrix ngm=new Matrix(3,3);
      ngm.set(0, 0, Math.min(Math.min(tpfcv.fpl,popfcv.fpl), ngfcv.fpm));
      ngm.set(0, 1, Math.min(Math.min(tpfcv.fpl,popfcv.fpm), ngfcv.fpm));
      ngm.set(0, 2, Math.min(Math.min(tpfcv.fpl,popfcv.fph), ngfcv.fpm));

      ngm.set(1, 0, Math.min(Math.min(tpfcv.fpm,popfcv.fpl), ngfcv.fpm));
      ngm.set(1, 1, Math.min(Math.min(tpfcv.fpm,popfcv.fpm), ngfcv.fpm));
      ngm.set(1, 2, Math.min(Math.min(tpfcv.fpm,popfcv.fph), ngfcv.fpm));

      ngm.set(2, 0, Math.min(Math.min(tpfcv.fph,popfcv.fpl), ngfcv.fpm));
      ngm.set(2, 1, Math.min(Math.min(tpfcv.fph,popfcv.fpm), ngfcv.fpm));
      ngm.set(2, 2, Math.min(Math.min(tpfcv.fph,popfcv.fph), ngfcv.fpm));

      Matrix ngh=new Matrix(3,3);
      ngh.set(0, 0, Math.min(Math.min(tpfcv.fpl,popfcv.fpl), ngfcv.fph));
      ngh.set(0, 1, Math.min(Math.min(tpfcv.fpl,popfcv.fpm), ngfcv.fph));
      ngh.set(0, 2, Math.min(Math.min(tpfcv.fpl,popfcv.fph), ngfcv.fph));

      ngh.set(1, 0, Math.min(Math.min(tpfcv.fpm,popfcv.fpl), ngfcv.fph));
      ngh.set(1, 1, Math.min(Math.min(tpfcv.fpm,popfcv.fpm), ngfcv.fph));
      ngh.set(1, 2, Math.min(Math.min(tpfcv.fpm,popfcv.fph), ngfcv.fph));

      ngh.set(2, 0, Math.min(Math.min(tpfcv.fph,popfcv.fpl), ngfcv.fph));
      ngh.set(2, 1, Math.min(Math.min(tpfcv.fph,popfcv.fpm), ngfcv.fph));
      ngh.set(2, 2, Math.min(Math.min(tpfcv.fph,popfcv.fph), ngfcv.fph));
```

```
      double vlc=Math.max(Math.max(ngl.get(0, 0), ngl.get(0, 1)), Math.max(
          ngm.get(0, 0), ngm.get(0, 1)));
      vlc=Math.max(vlc, ngh.get(0, 0));
      double lc=Math.max(ngl.get(0, 2), ngl.get(1, 0));
      lc=Math.max(lc, Math.max(ngm.get(0, 2), Math.max(ngh.get(0, 1), ngh.get
          (0, 2))));
      double mc = Math.max(ngl.get(1, 1), ngl.get(1, 2));
      mc=Math.max(mc, Math.max(Math.max(ngm.get(1, 0), ngm.get(1, 1)), ngm.
          get(1, 2)));
      mc=Math.max(mc, Math.max(ngh.get(1, 0), ngm.get(1, 1)));
      double hc=Math.max(Math.max(ngl.get(2, 0), ngl.get(2, 1)), ngm.get(2,
          0));
      hc=Math.max(hc, Math.max(ngh.get(1, 2), ngh.get(2, 0)));
      double vhc=Math.max(Math.max(ngl.get(2, 2), ngm.get(2, 1)), ngm.get(2,
          2));
      vhc=Math.max(vhc, Math.max(ngh.get(2, 1), ngh.get(2, 2)));

      double x= (vlc*0.0 + lc * 0.3 + mc * 0.5 + hc * 0.7 + vhc * 1)/ (vlc+lc
          +mc+hc+vhc);

      fuzzyprob.set(i, j,x);
    }
  }
  }
}

/**
 * Floating Index functions
 */

private void updateNodeIndexCache(NodeIndexRecord qir){
  boolean found=false;
  for( int i=0;i<indexCache.size();i++){
    if (indexCache.get(i).n.equals(qir.n) && indexCache.get(i).rno==qir.rno){
      found=true;
      if(qir.cdate>=indexCache.get(i).cdate) indexCache.set(i,qir);
    }
    }
  if(!found)
    indexCache.add(qir);
}
private void updateNodeIndexCache(QueryIndexRecord qir){
  boolean found=false;
  for( int i=0;i<indexCache.size();i++){
    if (indexCache.get(i).n.equals(qir.n) && indexCache.get(i).rno==qir.rno){
      found=true;
      if(qir.cdate>=indexCache.get(i).cdate){
        indexCache.set(i,new NodeIndexRecord(qir));
      }
    }
    }
  if(!found)
    indexCache.add(new NodeIndexRecord(qir));
}
public void addIndexCacheToQuery(Query m){
  m.qc.clear();
  Collections.sort(indexCache, new Comparator<NodeIndexRecord>(){
    public int compare(NodeIndexRecord o1, NodeIndexRecord o2){
      if(o1.lldate<o2.lldate) return -1;
      if(o1.lldate>o2.lldate) return 1;
      return 0;
    }});
  int size=0;
  for( int i=0;i<indexCache.size();i++){
      indexCache.get(i).lldate=new Date().getTime()/1000;
      m.qc.add(new QueryIndexRecord(indexCache.get(i)));
      size=size+Constants.BF_QIRSIZE;
      if(size>Constants.BF_QUERYSIZE) break;
    }
}

ArrayList<Node> searchIndexCache(KeyWord k)
{
  ArrayList<Node> al=new ArrayList<Node>();
  for(int i=0;i<indexCache.size();i++)
  {
    if(indexCache.get(i).bf.hasKey(k.kw))
      al.add(indexCache.get(i).n);
  }
  return al;
}

void refreshIndexCacheforLocalItems(){
  for(int i=0;i<indexCache.size();i++){
    if(indexCache.get(i).n.nodeno==this.nodeno) indexCache.remove(i);
  }
  QueryIndexRecord q=new QueryIndexRecord(); //assumtpion that no of items <
      Constants.BF_NOOFELEMENTS
  q.B=1;
  q.cdate=new Date().getTime()/1000;
  q.n=this;
  q.rno=1;
  int size=0;
  int k=1;
  for(int i=0;i<ic.size();i++){
    q.bf.put(ic.get(i).kw.kw);
    size=size+1;
    if(size>=Constants.BF_NOOFELEMENTS){
      updateNodeIndexCache(q);
      q= new QueryIndexRecord();
      q.B=1;
      q.cdate=new Date().getTime()/1000;
      q.n=this;
      q.rno=k++;
      size=0;
    }
    }
  updateNodeIndexCache(q);
}

void updateIndexcacheFromQuery(Message m){
    Query qy=(Query)m;

    for(int i=0;i<qy.qc.size();i++){
      boolean found=false;
        for(int j=0; j<indexCache.size();j++){
          if(qy.qc.get(i).n.nodeno==indexCache.get(j).n.nodeno && qy.qc.get(i
              ).rno==indexCache.get(j).rno){
            found=true;
            if(qy.qc.get(i).cdate>=indexCache.get(j).cdate){
              indexCache.remove(j);
              updateNodeIndexCache(qy.qc.get(i));
      }
     }
  }
  if(!found)
   updateNodeIndexCache(qy.qc.get(i));
 }
}

/**
 * QoS utility functions
 */
double getQuickFindProbability(int ngindex, KeyWord kw) {
  double prob = 0;
  for (int i = 0; i < Constants.NO_OF_BLOOMFILTERS; i++)
  if (ngbf[ngindex][i].hasKey(kw.kw)) {
    prob = prob + bf_levelprob[i];
   }
  return prob;
 }

 int getNormalizedBW(float bw) {
  double max = Constants.MAX_BW;
  if (bw >= 0 && bw <= max / 10)
   return 10;
  if (bw > max / 10 && bw <= 2 * max / 10)
   return 9;
  if (bw > 2 * max / 10 && bw <= 3 * max / 10)
   return 8;
  if (bw > 3 * max / 10 && bw <= 4 * max / 10)
   return 7;
  if (bw > 4 * max / 10 && bw <= 5 * max / 10)
   return 6;
  if (bw > 5 * max / 10 && bw <= 6 * max / 10)
   return 5;
  if (bw > 6 * max / 10 && bw <= 7 * max / 10)
   return 4;
  if (bw > 7 * max / 10 && bw <= 8 * max / 10)
   return 3;
  if (bw > 8 * max / 10 && bw <= 9 * max / 10)
   return 2;
  if (bw > 9 * max / 10 && bw <= 10 * max / 10)
   return 1;
  return 0;
 }

 int getNormalizedQFProb(double quickfindprob) {
  double max = 0.875;
  if (quickfindprob >= 0 && quickfindprob <= max / 10)
   return 10;
  if (quickfindprob > max / 10 && quickfindprob <= 2 * max / 10)
   return 9;
  if (quickfindprob > 2 * max / 10 && quickfindprob <= 3 * max / 10)
   return 8;
  if (quickfindprob > 3 * max / 10 && quickfindprob <= 4 * max / 10)
   return 7;
  if (quickfindprob > 4 * max / 10 && quickfindprob <= 5 * max / 10)
   return 6;
  if (quickfindprob > 5 * max / 10 && quickfindprob <= 6 * max / 10)
   return 5;
  if (quickfindprob > 6 * max / 10 && quickfindprob <= 7 * max / 10)
   return 4;
  if (quickfindprob > 7 * max / 10 && quickfindprob <= 8 * max / 10)
    return 3;
  if (quickfindprob > 8 * max / 10 && quickfindprob <= 9 * max / 10)
    return 2;
  if (quickfindprob > 9 * max / 10 && quickfindprob <= 10 * max / 10)
    return 1;
  return 0;
}

int getNormalizedDelay(float delay) {
  double max = Constants.MAX_DELAY;
  if (delay >= 0 && delay <= max / 10)
    return 1;
  if (delay > max / 10 && delay <= 2 * max / 10)
    return 2;
  if (delay > 2 * max / 10 && delay <= 3 * max / 10)
    return 3;
  if (delay > 3 * max / 10 && delay <= 4 * max / 10)
    return 4;
  if (delay > 4 * max / 10 && delay <= 5 * max / 10)
    return 5;
  if (delay > 5 * max / 10 && delay <= 6 * max / 10)
    return 6;
  if (delay > 6 * max / 10 && delay <= 7 * max / 10)
    return 7;
  if (delay > 7 * max / 10 && delay <= 8 * max / 10)
    return 8;
  if (delay > 8 * max / 10 && delay <= 9 * max / 10)
    return 9;
  if (delay > 9 * max / 10 && delay <= 10 * max / 10)
    return 10;
  return 0;
}

int getNormalizedAccDelay(float delay) {
  double max = 6000;
  if (delay >= 0 && delay <= max / 10)
    return 1;
  if (delay > max / 10 && delay <= 2 * max / 10)
    return 2;
  if (delay > 2 * max / 10 && delay <= 3 * max / 10)
    return 3;
  if (delay > 3 * max / 10 && delay <= 4 * max / 10)
    return 4;
  if (delay > 4 * max / 10 && delay <= 5 * max / 10)
    return 5;
  if (delay > 5 * max / 10 && delay <= 6 * max / 10)
    return 6;
  if (delay > 6 * max / 10 && delay <= 7 * max / 10)
    return 7;
  if (delay > 7 * max / 10 && delay <= 8 * max / 10)
    return 8;
  if (delay > 8 * max / 10 && delay <= 9 * max / 10)
    return 9;
  if (delay > 9 * max / 10 && delay <= 10 * max / 10)
    return 10;
  return 0;
}


double getWeightedAvgCost(int i, Message m) {
  double cost = m.bw_weight * this.getNormalizedBW(bw[i])
      + m.delay_weight * this.getNormalizedAccDelay(m.delay_accumulated+dl[
          i])
      + m.quckfind_weight * this.getNormalizedQFProb(qfp[i]);
  return cost;
}

double getCompromiseProgCost(int i, Message m) {
  double idealbw, idealdl;
  double idealqfp;
  double diffbw, diffdl;
  double diffqfp;
  double bwa, dla, qfpa;
  idealbw = Utils.getmax(bw, ng.size());
  idealdl = Utils.getmin(dl, ng.size());
  idealqfp = Utils.getmaxf(qfp, ng.size());
  diffbw = Utils.getmax(bw, ng.size()) - Utils.getmin(bw, ng.size());
  diffdl = Utils.getmax(dl, ng.size()) - Utils.getmin(dl, ng.size());
  diffqfp = Utils.getmaxf(qfp, ng.size()) - Utils.getminf(qfp, ng.size());
  bwa = bw[i];
  dla = -1 * dl[i];
  qfpa = qfp[i];
  if (diffbw == 0 || diffdl == 0 || diffqfp == 0)
    return 0;
  double cost = m.bw_weight * Math.abs(((idealbw - bwa) / diffbw))
      + m.delay_weight * Math.abs(((idealdl - dla) / diffdl))
      + m.quckfind_weight * Math.abs(((idealqfp - qfpa) / diffqfp));
  return cost;
}

double[] TOPSIS_init(Message m) {
  double idealbw, idealdl;
  double idealqfp;
  double nidealbw, nidealdl, nidealqfp;

  double bw1[] = new double[ng.size()];
  double dl1[] = new double[ng.size()];
  double qfp1[] = new double[ng.size()];

  double dplus[] = new double[ng.size()];
  double dminus[] = new double[ng.size()];
  double cost[] = new double[ng.size()];

  for (int i = 0; i < ng.size(); i++)
    bw1[i] = getNormalizedBW(bw[i]);
  //bw1 = Utils.normalize_method4(bw1);

  for (int i = 0; i < ng.size(); i++)
    bw1[i] = bw1[i] * m.bw_weight;

  for (int i = 0; i < ng.size(); i++)
    dl1[i] = getNormalizedDelay(dl[i]);
  //dl1 = Utils.normalize_method4(dl1);
  for (int i = 0; i < ng.size(); i++)
    dl1[i] = dl1[i] * m.delay_weight;

  for (int i = 0; i < ng.size(); i++)
    qfp1[i] = getNormalizedQFProb(qfp[i]);
  //qfp1 = Utils.normalize_method4(qfp1);
  for (int i = 0; i < ng.size(); i++)
    qfp1[i] = qfp1[i] * m.quckfind_weight;
  idealbw = Utils.getmaxf(bw1, ng.size());
  nidealbw = Utils.getminf(bw1, ng.size());

  nidealdl = Utils.getminf(dl1, ng.size());
  idealdl = Utils.getmaxf(dl1, ng.size());

  idealqfp = Utils.getmaxf(qfp1, ng.size());
  nidealqfp = Utils.getminf(qfp1, ng.size());

  for (int i = 0; i < ng.size(); i++) {
    dplus[i] = Math.sqrt((bw1[i] - idealbw) * (bw1[i] - idealbw)
        + (dl1[i] - idealdl) * (dl1[i] - idealdl)
        + (qfp1[i] - idealqfp) * (qfp1[i] - idealqfp));
    dminus[i] = Math.sqrt((bw1[i] - nidealbw) * (bw1[i] - nidealbw)
        + (dl1[i] - nidealdl) * (dl1[i] - nidealdl)
        + (qfp1[i] - nidealqfp) * (qfp1[i] - nidealqfp));

    if (dminus[i] == 0 && dplus[i] == 0)
      cost[i] = 0;
    else
      cost[i] = dminus[i] / (dplus[i] + dminus[i]);
  }

  return cost;
}
```

A.4 SybilNode Implementation

The following is the implementation of SybilNode class in PlanteSim for simulating the method described in section 5.3.

Listing A.4: SybilNode

```
package planet.chord;

import java.util.Vector;
import planet.chord.message.BroadcastMessage;
import planet.chord.message.IdMessage;
import planet.chord.message.NodeMessage;
import planet.chord.message.SuccListMessage;
import planet.commonapi.EndPoint;
import planet.commonapi.Id;
import planet.commonapi.Message;
import planet.commonapi.NodeHandle;
import planet.commonapi.RouteMessage;
import planet.commonapi.exception.InitializationException;
import planet.generic.commonapi.factory.GenericFactory;
import planet.simulate.Globals;
import planet.simulate.Logger;
import planet.simulate.MessageListener;
import planet.simulate.Results;

public class SybilNode extends ChordNode
{
  public static planet.commonapi.Message[] storage=new planet.commonapi.
      Message[10];
```

```
public SybilNode() throws InitializationException{
  super();
}
public void dispatcher(RouteMessage msg) {
  //Response message but not successor list type (key == null)
  if (msg.getMode() == REPLY && msg.getKey() != null) {
    String key = msg.getKey();
    try {
      ((MessageListener) listeners.get(key)).onMessage(msg);
    } catch (NullPointerException e) {
      Logger.log(        + id +
          + msg.getKey() +    , Logger.ERROR_LOG);
    }
    removeMessageListener(key);
  } else if (msg.getMode() == Globals.ERROR) {
    if (msg.getKey() != null) {
      String key_fp = msg.getKey();
      Logger.log(        + this.id +                        + key_fp
          +          + Globals.typeToString(msg.getType()) +
          + msg.getMessage(), Logger.MSG_LOG);
      MessageListener lst = (MessageListener) listeners.get(key_fp);
      if (lst != null) {
        removeMessageListener(key_fp);
      }
    }
    //Successor lost
    if (finger[0] != null && msg.getSource().equals(finger[0])
        && succList.size() > 0) {
      //if not exists, succ_list is unchanged
      succList.remove(finger[0]);
      if (succList.size() > 0) {
        finger[0] = (NodeHandle) succList.firstElement();
        //send notify
                  this.sendMessage(msg,null,nodeHandle,finger[0],finger[0],
                      SET_PRE,REFRESH,new NodeMessage(nodeHandle));
      }
    } else
      //if source not exists, succ_list is unchanged
      succList.remove(msg.getSource());
  } else {
    switch (msg.getType()) {
      //DATA
      case DATA :
          dispatchDataMessage(msg,REQUEST,REFRESH);
        break;
      //CONTROL
      //REFRESH
      case SET_SUCC :
        NodeHandle succ = ((NodeMessage) msg.getMessage()).getNode();
        setSucc(succ);
        GenericFactory.freeMessage(msg);
        break;
      case SET_PRE :
        setPred(((NodeMessage) msg.getMessage()).getNode());
        GenericFactory.freeMessage(msg);
        break;
      case NOTIFY :
        notify(msg.getSource());
        GenericFactory.freeMessage(msg);
        break;
      case BROADCAST :
```

```
  NodeHandle r, new_limit;
  BroadcastMessage bm = (BroadcastMessage) msg.getMessage();
  NodeHandle limit = bm.getLimit();
  Id limitId = limit.getId();
  planet.commonapi.Message info = bm.getInfo();

  for (int i = 0; i < bitsPerKey - 1; i++) {
    //Skip a redundant finger
    if (!finger[i].equals(finger[i + 1])) {
      //Forward while within "Limit"
      if (finger[i].getId().between(this.id, limitId)) {
        r = finger[i];
        //New Limit must not exceed Limit
        if (finger[i + 1].getId().between(this.id, limitId)) {
          new_limit = finger[i + 1];
        } else {
          new_limit = limit;
        }
        //no reuse of RouteMessage msg ==> send one
        // message to different nodes ==> requires
        // different messages
        planet.commonapi.RouteMessage aMsg = null;
        try {
          // String appId,Id from, Id to, Id nextHop
          aMsg = getBroadcastMessage(msg
              .getApplicationId(), this.nodeHandle, r, r,
              new BroadcastMessage(info,new_limit));
          sendMessage(aMsg);
          Results.incTraffic();
        } catch (InitializationException e) {
          Logger.log(

              + e.getMessage(), Logger.ERROR_LOG);
        }
      }
    }
  }
  Logger.log(                    + this.id +
      + info, Logger.EVENT_LOG);
  msg.setMessage(info);
  Results.decTraffic();
  ((EndPoint) endpoints.get(msg.getApplicationId())).scheduleMessage(
      msg, 0);
  break;
//REQUEST
case FIND_SUCC :
  //source --> join();
  NodeHandle fSucc = findSuccessor(msg.getSource());
  if (fSucc != null) {
                this.sendMessage(msg,msg.getKey(),nodeHandle,msg.
                    getSource(),msg.getSource(),msg.getType(),REPLY,
                    new NodeMessage(fSucc));
  } else {
    String key_fp = GenericFactory.generateKey();
                NodeHandle aux = closestPrecedingFinger(msg.getSource
                    ().getId());
                addMessageListener(key_fp, new FindPredListener(this,
                     msg.getKey()));
                this.sendMessage(msg,key_fp,nodeHandle,aux,aux,
                    FIND_PRE,REQUEST,new IdMessage(msg.getSource().
                    getId()));
  }
  break;
case FIND_PRE :
    Id idMesg = ((IdMessage) msg.getMessage()).getNode();
  if (finger[0] != null && idMesg.betweenE(this.id, finger[0].getId()
      )) {
    //return successor
    Id msgId = ((IdMessage) msg.getMessage()).getNode();
    try {
                    sendMessage(msg,msg.getKey(),GenericFactory.
                        buildNodeHandle(msgId,true),
                            msg.getSource(),msg.getSource(),FIND_PRE,
                                REPLY,new NodeMessage(getSucc()));
    } catch (InitializationException e1) {
      e1.printStackTrace();
    }
   } else if (idMesg.equals(getId())) {
      try {
                        sendMessage(msg,msg.getKey(),GenericFactory.
                            buildNodeHandle(idMesg, true),
                                msg.getSource(),msg.getSource(),
                                    FIND_PRE,REPLY,new NodeMessage(
                                    nodeHandle));
      } catch(InitializationException e) {
        e.printStackTrace();
      }
   } else {
    //next node
                NodeHandle aux = closestPrecedingFinger(idMesg);
                sendMessage(msg,msg.getKey(),msg.getSource(),aux,aux,
                    msg.getType(),msg.getMode(),msg.getMessage());
  }
  break;
case GET_PRE :
  //origen --> stabilize();
            sendMessage(msg,msg.getKey(),nodeHandle,msg.getSource(),
                msg.getSource(),GET_PRE,REPLY,new NodeMessage(
                predecessor));
  break;
case SUCC_LIST :
  if (msg.getMode() == REQUEST) {
                this.sendMessage(msg,msg.getKey(),nodeHandle,msg.
                    getSource(),msg.getSource(),SUCC_LIST,REPLY,new
                    SuccListMessage(succList));
  } else if (msg.getMode() == REPLY) {
    SuccListMessage succs = (SuccListMessage) msg.getMessage();
                succList.clear();
                succList.add(msg.getSource());
                succList.addAll(succs.getSuccs());
                cleanSuccList();
    GenericFactory.freeMessage(msg);
  }
case OWNER: //added
  Vector succlist = this.succList;
  for(int i=0; i<succlist.size();i++)
  {
    String skey = GenericFactory.generateKey();
      this.sendMessage(skey,nodeHandle,(NodeHandle)succlist.get(i),
          REPLICATE,REQUEST,(Message)msg);
  }
  break;
```

```
        case REPLICATE: //added
          for(int i=0;i<10;i++)
          {
              if(storage[i] == null)
                continue;
              else
                storage[i]=msg.getMessage();
                break;
          }
          break;

      }
    }
  }

}
```

Appendix B

Psychometric Tests

Our approach for detecting Sybils using Psychometric tests is presented in chapter 5. The following sections present some graphs, questionnaire and DBSCAN algorithm implementaion.

B.1 MBTI Questionnaire

The following are the questions designed for MBTI test. There are 36 questions

```
Q1. In interacting with friends and colleagues:
a.  I tend to talk on all topics whether I know them or not
b.  I like to talk on the topics i know somewhat but prefer to hear about
    those which are completely unknown to me
c.  i don't have any idea on how I interact with friends
d.  Mostly I keep hearing others and restrict myself to talk only on topics i
     am very much familiar with
e.  I keep hearing even if i am very much well-versed in that topic

Q2. If you have to do unfamiliar things, you prefer :
a.  step-by-step Directions.
b.  to figure out with considerable help
c.  doing in random ways
d.  to figure out with a little help.
e.  to figure out myself without any help

Q3. I often :
a.  Do things first and then start thinking
b.  Start doing things after a bit of thinking.
c.  I believe doing things is not related to thinking
d.  Think thoroughly of most weighted factors before doing
e.  Think thoroughly of all the factors in detail before doing

Q4. If you have to decide :
a. I take all my decisions right away.
b. I take most of my decisions immediately, but keep some options open.
c. never took decisions
d. I take a few decisions, but keep most of the options open.
e. Never. I'd like to keep options open.

Q5. Do you think out loud i.e. speak out while you think:
a.  Yes, I think out loud all the time
b.  Usually , i think out loud
c.  Thinking! I never did that
d.  No, most of the time i think quietly inside my head.
```

e. Never. I always think quietly inside my head

Q6. 6. You can described as a person who thinks :
a. Always about possibilities in future
b. Most of the time about future possibilities and less about present.
c. I have no idea about both present and future. I live with myself.
d. More about present and less about future
e. Always about present.

Q7. Do you always see faults :
a. Yes, i see faults with everything.
b. Yes, i see faults with things/people i closely observe/work with.
c. i see life as full of faults at the same time life is faultless.
d. I tend to see no faults but when I am asked to, I do find faults
e. No, i generally don’t see faults in anything even if I am asked to.

Q8. Are you comfortable with schedules :
a. Yes, I’m always comfortable with schedules
b. Yes, usually I’m comfortable with schedules
c. Never had a schedule
d. Somewhat, More comfortable in being spontaneous
e. No, I want complete freedom to be spontaneous.

Q9. Do you prefer compassion to Honesty and directness?
a. I always go beyond my limits to help anyone in problem
b. I go beyond rules and my level to help only if it doesn’t implicate me
c. Never helped anyone
d. I prefer to stay in rules and my limits to help others but go beyond for near and dear ones
e. I always stay in rules and my limits in helping others

Q10. When you do something do you think about its future implications :
a. Yes, I always think about future implications
b. Yes , i usually think about future implications
c. Never considered any implication in life.
d. No , most of the time i think about its immediate implications.
e. No, I think about immediate implications

Q11. Do you pay attention to time and be prompt:
a. Yes, i always pay attention to time and I’m always prompt.
b. I pay attention to time almost every time.
c. No idea
d. No, most of the time i run late.
e. No , i always run late

Q12. When you have to judge an issue that affects someone, you are more convinced by :
1. only rational arguments
2. to a great extent rational arguments but feelings of the concerned do matter
3. never judged anyone in life
4. feelings and emotions of the person comes first and arguments are considered later
5. only the feelings of the person

Q13. Are your decisions based on Values and feelings :
a. My decisions are always based on values and feelings
b. Usually, my decisions are based on values and feelings
c. I don’t understand values and feelings
d. yes , most of my decisions are objective
e. No , All my decisions are always objective

Q14. Are you comfortable being alone?
a. I'm always comfortable being alone.
b. Usually , I'm comfortable alone but i do need people sometimes
c. No idea
d. I like to be with people, but not always
e. I like to be with people always

Q15. Regarding work, select the best that suits you :
1. Energetic and passionate to do any task
2. i am very passionate and energetic to do things that give me high return
3. I never worked
4. tasks that give me content, i carry them with determination irrespective of monetary gain
5. any task given to me, i carry it with determination even if a higher return alternative is available

Q16. What do you think about general rules :
a. I always think all the rules are necessary.
b. I always think that most of the rules are necessary but i question a few.
c. Rules don't exist for me.
d. I think most of the rules are unnecessary and a few as necessary.
e. I question the need for most rules

Q17. Do you prefer to work at a steady pace :
a. Yes, i always work at a steady pace
b. I do most of my work at a steady pace
c. Never worked
d. I work at a steady pace only when it is required.
e. No, i prefer to work in bursts of energy.

Q18. Are you easily distracted by allurements and disruptions :
a. Yes, always distracted by any issue.
b. Easily distracted by issues that doesn't concern me
c. No idea
d. Can withstand the issues that don't concern me.
e. No , I have strong powers of concentration

Q19. You are best described as :
a. Always honest.
b. I want to be honest, but sometimes the situation forces me to be diplomatic.
c. Dishonest, non-diplomatic person
d. Mostly diplomatic but honest with some people.
e. always diplomatic and tactful

Q20. Do you prefer to develop components for your project :
a. Yes, i always prefer building/developing all components on my own.
b. I build/develop most of the parts, but use available components for critical things
c. Never did a project in life
d. Yes, only if they are not readily available.
e. No, i prefer using available components only.

Q21. Do you make all your decisions?
1. Yes, i make all my decisions.
2. Yes, most of the time i make my own decisions.
3. never made a decision in life
4. most of the time it is difficult to make decisions
5. always, I have difficulty in making decisions

Q22. Do you prefer to start new projects :
a. Yes , i am always excited about new projects.
b. Yes, but sometimes it depends on my current project.
c. Never got exposure to a project
d. Only if my current projects permit the time
e. i prefer completing already assigned projects

Q23. Do you like arguing all the time :
a. Yes, arguing is fun.
b. Usually, i find myself in an argument.
c. No idea
d. Only when i am obliged to.
e. No , i like to avoid such things

Q24. Do you like to focus on details or Big Picture of a project:
a. I focus more on the details.
b. I focus more on the details with a little idea of the big picture.
c. Never worked on project
d. I generally focus on the big picture but some details are necessary.
e. Focus entirely on big picture.

Q25. You are motivated by :
1. Achieving the goal irrespective of criticism from all people
2. Achieving the goal with some people supporting it by appreciating
3. Can't say what motivates me. I am dull to motivation
4. People appreciating me to pursue the goal
5. Unless i receive appreciation from all people, i can't work to achieve the goal

Q26. How can you describe your interaction with people :
a. i am at ease and enthusiastic in talking to everyone
b. i am at ease and enthusiastic in talking to most of the people but keep myself reserved with some
c. i don't know how i talk. i just talk.
d. most of the time i keep myself reserved and speak to myself. Only with few people i talk freely
e. i am always reserved and speak to myself.

Q27. While interacting with others you see yourself as :
a. Cool and reserved with everyone
b. Cool and reserved with most of the people, but warm and friendly with some.
c. I don't know about how i project myself.
d. Warm and friendly with most of the people, but cool and reserved with some.
e. Very warm and friendly with everyone.

Q28. If you are asked to judge some ideas for a project, what do you prefer :
a. I like solutions that are implementable and they need not be creative.
b. I prefer solutions that are implementable but should be creative in some aspects.
c. No idea what a project is
d. creative Ideas , but they should be partially implementable
e. creative in all aspects, but need not be practical

Q29. Do you like to notice everything/issue with a different perspective :
a. Yes, I like to see everything differently.
b. Yes, I like to see most of the things differently.
c. No idea
d. No , I like to see most of the things in a conventional way

```
e.  No, I rather like to notice and remember facts.

Q30.  If someone says something bad about you, do you take it personally?:
a.  No, i don’t bother what people say.
b.  Yes, only if it is someone who i care.
c.  No has told anything to me so far
d.  Yes, i tend take most of the things personally.
e.  Yes, i tend take everything personal.

Q31.  While working do you prefer a public role to being behind the scenes :
a.  Yes, in all of my work I prefer a public role.
b.  Yes, in most of my work i prefer a public role.
c.  I never had any work
d.  In most of my work, i prefer to work behind the scenes.
e.  Never, i like to work behind the scenes

Q32.  While working are you always serious :
a.  Yes, i’m always serious while working
b.  Usually, i’m serious while working
c.  I never got opportunity to work
d.  No , I try to be playful all the time , but sometimes i need to be
    serious.
e.  I’m always playful while working.

Q33.  If you are to work on a project with a fixed schedule, will you stick
    to it :
a.  Yes , i strictly adhere to the schedule
b.  Yes, i always stick to the schedule with a few exceptions
c.  I was never a part of project
d.  No, I prefer flexibility but sometimes I’m required to stick to the
    schedule
e.  No , I like being flexible. I can’t work on a fixed schedule.

Q34.  Do you believe in ’Play first, work later’?
a: Yes
b. Usually play first with small work periods in between.
c. I have no beliefs. I do things doubtfully.
d. Usually work first with small play-breaks in between.
e: No, I believe in  work first and play later.

Q35.  Do you trust your instincts?
a: No, I better believe in actual experience and logic, whether personal or
    of others.
b. I occasionally trust my instincts, but rely mostly on experience and logic
    .
c. I don’t trust anyone/anything.
d. I usually trust my instincts but occasionally rely on experience and logic
    .
e. Yes, always.

Q36.  Are you a:
a: Multi-tasker and prefer doing lot of things at a time
b. Usually a multi-tasker but tends to act as perfectionist from time to time
    .
c. I have no idea
d. Usually a perfectionist but tends to act as a multi-tasker from time to
    time.
e: Perfectionist and prefer to do one thing at a time.
```

B.2 Clustering Algorithm Implementation

Listing B.1: Source code for Point class and DBSCAN algorithm

```java
class Point {
  static double findDistance(Point p1, Point p2) {
    double sum = 0;
    for (int i = 0; i < p1.getDimension(); i++) {
      float x = Math.abs(p1.getCoordinate(i) - p2.getCoordinate(i));
      sum = sum + x * x;
    }
    return Math.sqrt(sum);
  }
  static long pid = 0;
  int d;
  float p[];
  boolean visited = false;
  boolean noise = false;
  String nick = null;
  int ecid;// expected cluster id
  int acid;// allocated cluster id
  static int primes[] = { 73, 179, 283, 419, 547, 661, 811, 947 };
  static int primes_colors[] = { 173, 281, 409, 541, 659, 809, 941, 1069 };
  long id;
  Point(int d) {
    this.d = d;
    p = new float[d];
    id = pid++;
    acid = -1;
  }
  double findSpearmansCoefficient(Point p1) {
    double sum = 0;
    for (int i = 0; i < d; i++) {
      sum = sum + (p[i] - p1.p[i]) * (p[i] - p1.p[i]);
    }
    return 1 - ((6 * sum) / (d * (d * d - 1)));
  }
  double findKindallsTauCoefficient(Point p1) {
    double Tau = 0;
    double xi, yi, xj, yj;
    int concord = 0;
    int discord = 0;
    for (int i = 0; i < d; i++) {
      xi = p[i];
      yi = p1.p[i];
      for (int j = i + 1; j < d; j++) {
        xj = p[j];
        yj = p1.p[j];
        if (xi > xj && yi > yj)
          concord++;
        if (xi < xj && yi < yj)
          concord++;
        if (xi > xj && yi < yj)
          discord++;
        if (xi < xj && yi > yj)
          discord++;
      }
    }
    Tau = (double) (concord - discord) / (double) (((d * (d - 1)) / 2));
    return Tau;
  }

  double findCosineSimilarity(Point p1) {
    double sum = 0;
    for (int i = 0; i < d; i++) {
      sum = sum +  p[i]
          * p1.p[i];
    }
    double dsum1 = 0;
    for (int i = 0; i < d; i++) {
      dsum1 = dsum1 + p[i]
          * (int) p[i];
    }
    dsum1 = Math.sqrt(dsum1);
    double dsum2 = 0;
    for (int i = 0; i < d; i++) {
      dsum2 = dsum2 +  p1.p[i]
          *  p1.p[i];
    }
    dsum2 = Math.sqrt(dsum2);

    return (sum / (dsum1 * dsum2));
  }

  double findPearsonCoefficient(Point p1) {
    double sumx = 0;
    double sumy = 0;
    double suma = 0, sumb = 0;
    double avgx, avgy;
    int d1 = d;
    for (int i = 0; i < d1; i++)
      sumx = sumx + p[i];
    avgx = sumx / d1;
    for (int i = 0; i < d1; i++)
      sumy = sumy +  p1.p[i];
    avgy = sumy / d1;

    for (int i = 0; i < d1; i++)
      suma = suma +  (p[i] - avgx)
          * ( p1.p[i] - avgy);

    for (int i = 0; i < d1; i++)
      sumx = sumx + ( p[i] - avgx)
          * (int) p[i] - avgx;
    for (int i = 0; i < d1; i++)
      sumy = sumy + ( p1.p[i] - avgy)
          * (int) p1.p[i] - avgy;

    sumb = Math.sqrt(sumx * sumy);
    if (suma / sumb > 0.9)
    return (suma / sumb);
  }

  float getCoordinate(int i) {
    return p[i];
  }

  int getDimension() {
    return d;
  }
```

```
 void print() {
  for(int i=0;i<d;i++)
   System.out.print(p[i]+  );
  System.out.print(  );
 }

}


void DBSCAN(ArrayList<Point> plist, double eps, int minpnts) {
  for (int i = 0; i < plist.size(); i++) {
   Point p = plist.get(i);
   if (p.visited == false) {
    p.visited = true;
    Cluster N = getNeighbors(p, i, eps);
    print(     + p.nick +     + N.size());
    if (N.size() < minpnts) {
     p.noise = true;
    }

    else {
     cluster_count++;
     Cluster c = new Cluster(cluster_count);
     clusters_found.add(cluster_count, c);
     expandCluster(p, i, N, c, eps, minpnts);
    }
   }
  }
 }

 void expandCluster(Point p, int index, Cluster N, Cluster c, double eps,
   int minpnts) {
  c.pnts.add(p);
  for (int i = 0; i < N.size(); i++) {
   Point p1 = N.pnts.get(i);
   if (p1.visited == false) {
    p1.visited = true;
    Cluster N1 = this.getNeighbors(p1, plist5.indexOf(p1), eps);
    if (N1.size() >= minpnts) {
     for (int j = 0; j < N1.size(); j++)
      N.pnts.add(N1.pnts.get(j));
    }
   }
   boolean found = false;
   for (int j = 0; j < clusters_found.size(); j++)
    for (int k = 0; k < clusters_found.get(j).size(); k++) {
     if (clusters_found.get(j).pnts.get(k).id == p1.id) {
      found = true;
      break;
     }
    }
   if (!found) {
    p1.acid = c.index;
    c.pnts.add(p1);

   }
  }

 }
```

B.3 Similarity Measures at Different Cluster Configurations

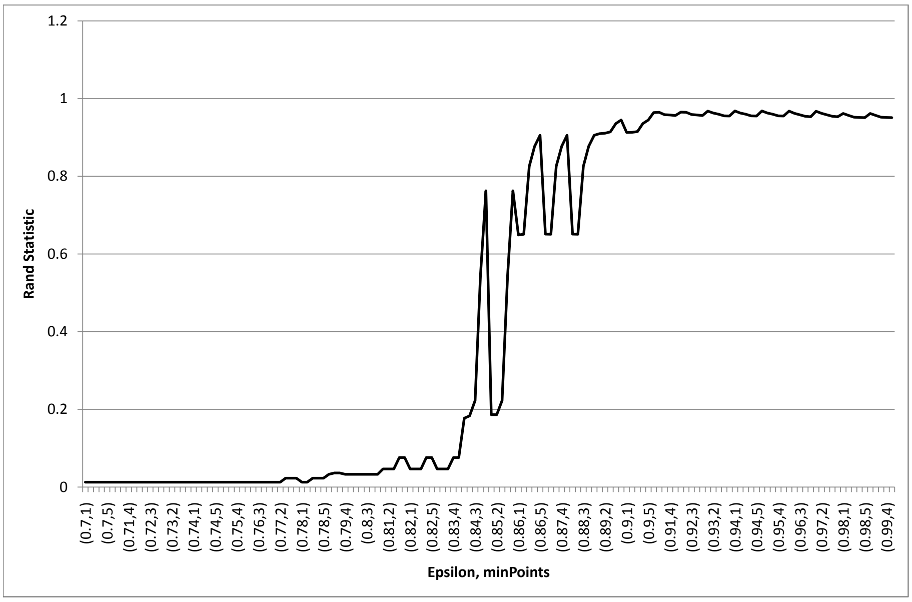


Figure B.1: Values of Rand Statistic for Spearman's ρ - Rand statistic is calculated for various combinations of ϵ and minPoints

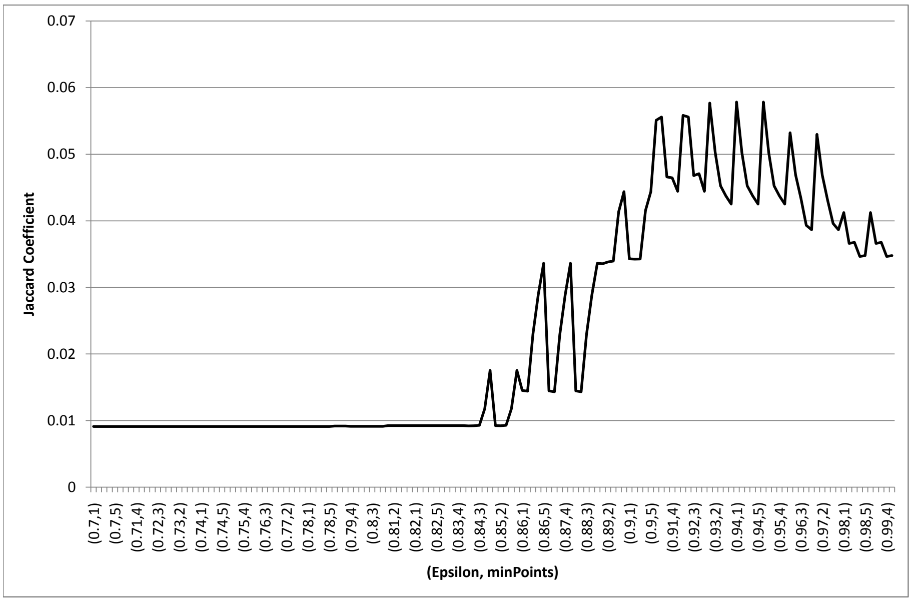


Figure B.2: Values of Jaccard Statistic for Spearman's ρ - Jaccard statistic is calculated for various combinations of ϵ and minPoints

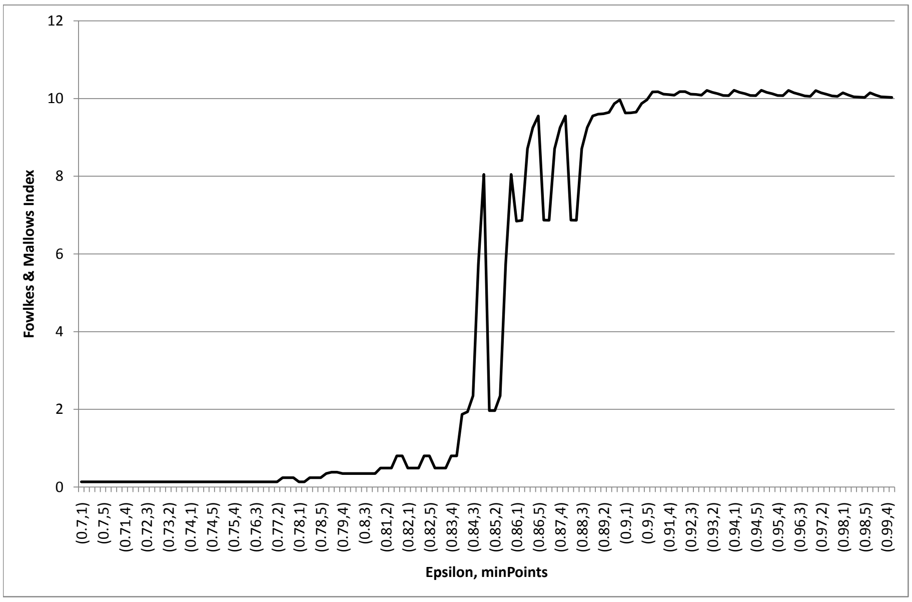


Figure B.3: Values of Fowlkes & Mallow's index for Spearman's ρ - Fowlkes & Mallow's index is calculated for various combinations of ϵ and minPoints

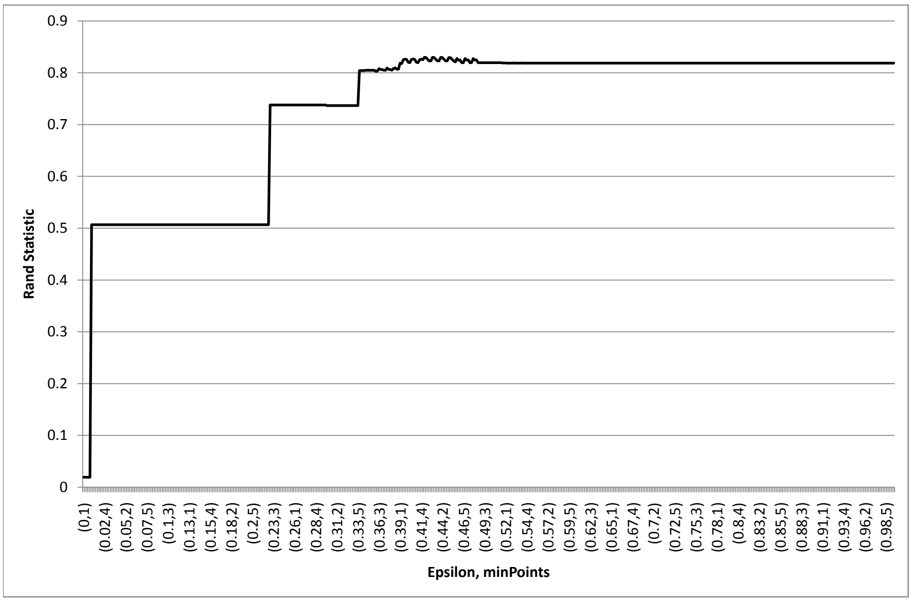


Figure B.4: Values of Rand Statistic for Pearson coefficient - Rand statistic is calculated for various combinations of ϵ and minPoints

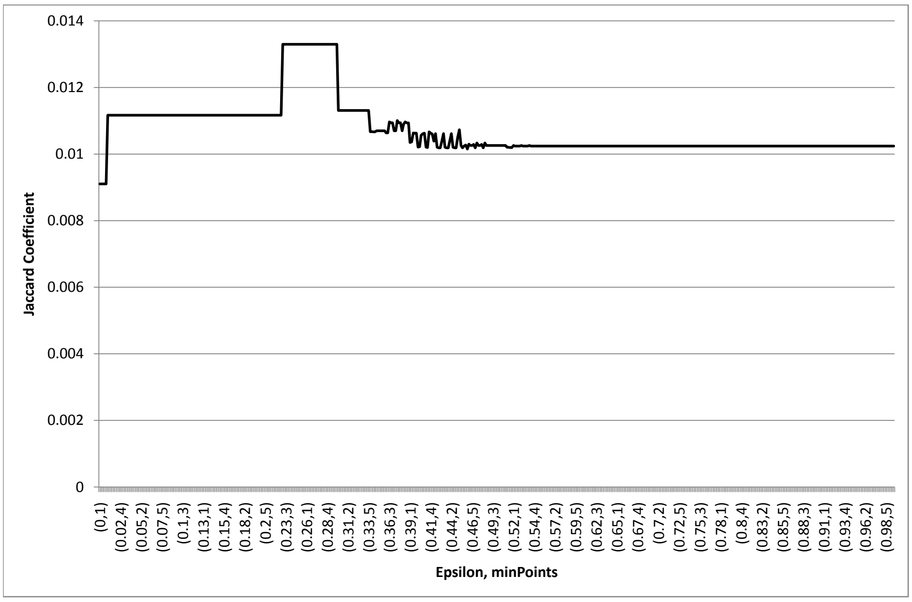


Figure B.5: Values of Jaccard Statistic for Pearson coefficient - Jaccard statistic is calculated for various combinations of ϵ and minPoints

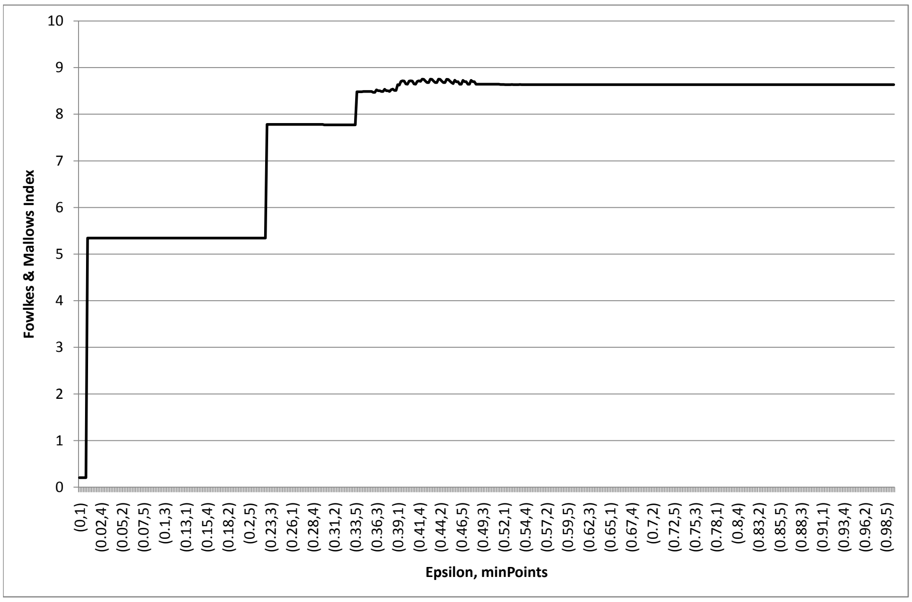


Figure B.6: Values of Fowlkes & Mallow's index for Pearson coefficient - Fowlkes & Mallow's index is calculated for various combinations of ϵ and minPoints

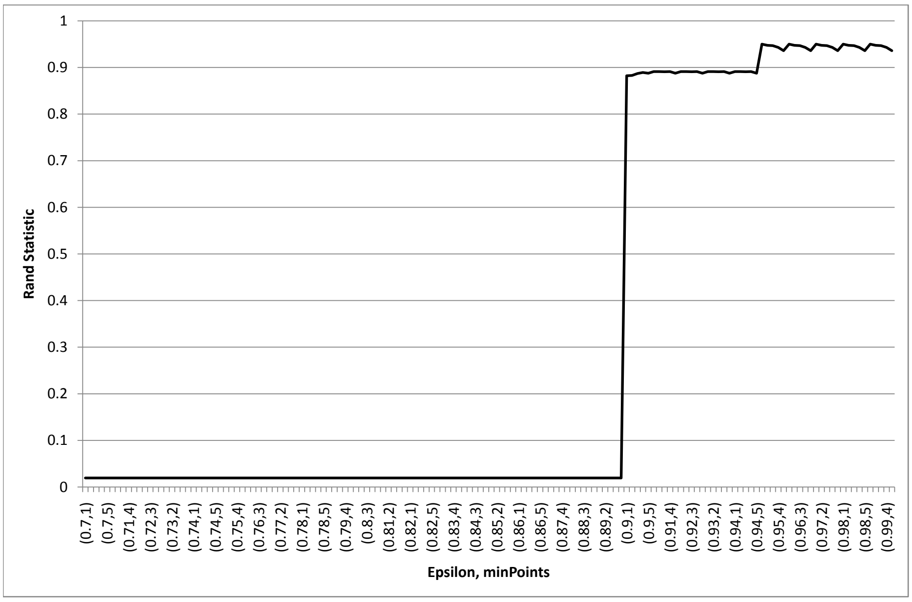


Figure B.7: Values of Rand Statistic for Cosine similarity coefficient - Rand statistic is calculated for various combinations of ϵ and minPoints

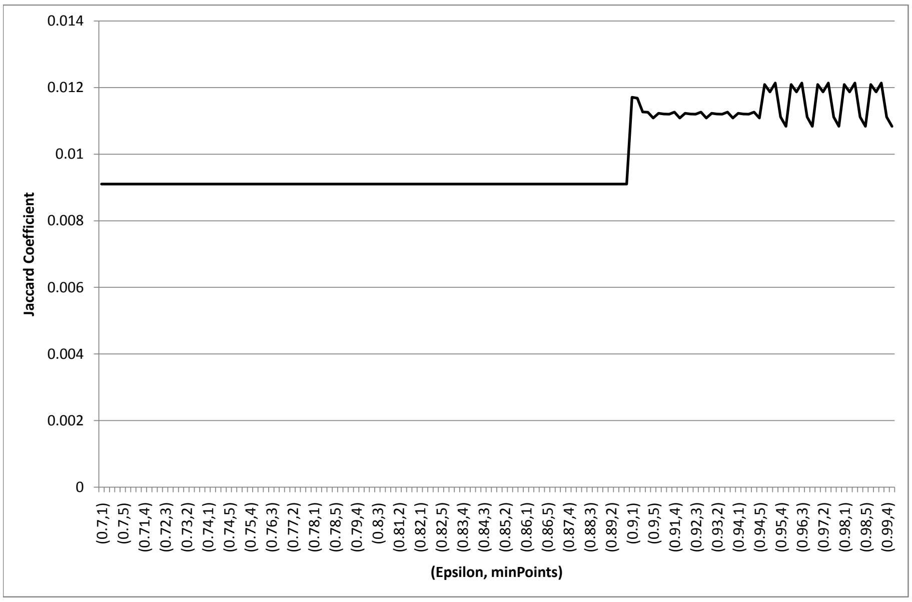


Figure B.8: Values of Jaccard Statistic for Cosine similarity coefficient - Jaccard statistic is calculated for various combinations of ϵ and minPoints

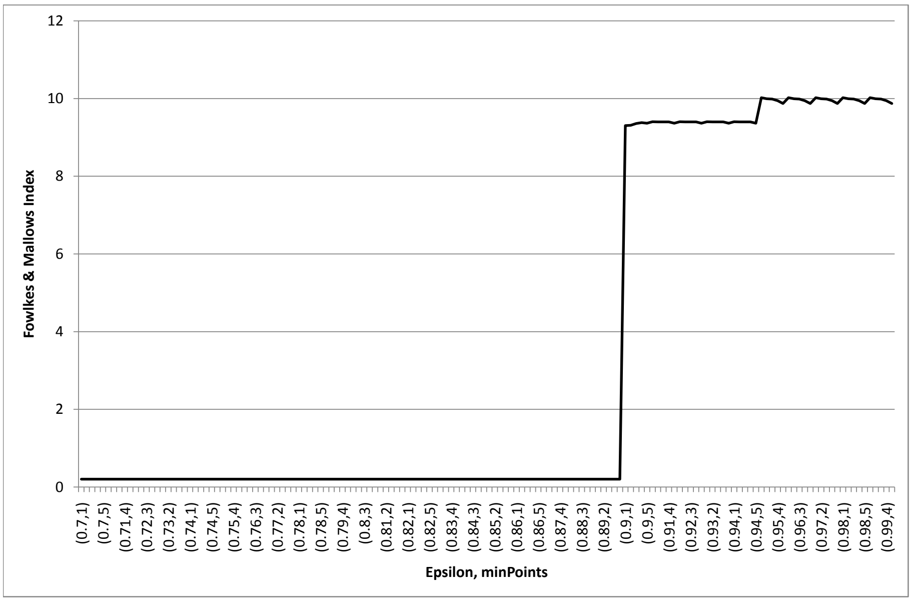


Figure B.9: Values of Fowlkes & Mallow's index for Cosine similarity coefficient - Fowlkes & Mallow's index is calculated for various combinations of ϵ and minPoints

References

[Adamic *et al.* 2001] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani and Bernardo A. Huberman. *Search in power-law networks*. Phys. Rev. E, vol. 64, no. 4, page 046135, Sep 2001. 27, 40, 44

[Adamic *et al.* 2003] Lada A. Adamic, Orkut Buyukkokten and Eytan Adar. *A social network caught in the Web.* First Monday, vol. 8, no. 6, 2003. 26

[Ahulló & López 2008] Jordi Pujol Ahulló and Pedro García López. *PlanetSim: an extensible framework for overlay network and services simulations*. In Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, Simutools '08, pages 45:1–45:1, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). 176

[Albert & Barabasi 1999] Jeong H. Albert R. and A. L Barabasi. *The Diameter of the World Wide Web*. Nature, vol. 401, pages 130–131, 1999. 26

[Albert *et al.* 2000] Reka Albert, Hawoong Jeong and Albert-Laszlo Barabasi. *Error and attack tolerance of complex networks*. NATURE, vol. 406, page 378, 2000. 28

[Amaral *et al.* 2000] L. A. N. Amaral, A. Scala, M. Barthélémy and H. E. Stanley. *Classes of small-world networks*. Proceedings of the National Academy of Sciences, vol. 97, no. 21, pages 11149–11152, October 2000. 26

[Androutsellis-theotokis & Spinellis 2004] Stephanos Androutsellis-theotokis and Diomidis Spinellis. *A survey of peer-to-peer content distribution technologies*. ACM Computing Surveys, vol. 36, pages 335–371, 2004. 3, 4

[Ang & Datta 2010] Rzadca K. Ang S. and A. Datta. *SharedMind: A tool for collaborative mind-mapping*. In IEEE International Conference on Multimedia and Expo (ICME). IEEE Computer Society, 2010. 8

[bab 2005] *Babelgum*. http://en.wikipedia.org/wiki/Babelgum, 2005. Retrieved on 6th March 2011. 9

[Baeza-Yates & Ribeiro-Neto 1999] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. Modern information retrieval. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. 17, 45, 90

[Balakrishnan *et al.* 2003] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris and Ion Stoica. *Looking up data in P2P systems*. Commun. ACM, vol. 46, no. 2, pages 43–48, February 2003. 37

[Bazzi & Konjevod 2005] Rida A. Bazzi and Goran Konjevod. *On the establishment of distinct identities in overlay networks*. In Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing, PODC '05, pages 312–320, New York, NY, USA, 2005. ACM. 52

[Bellman & Corporation 1957] R.E. Bellman and Rand Corporation. Dynamic programming. Rand Corporation research study. Princeton University Press, 1957. 192

[Bernstein *et al.* 2002] Philip A. Bernstein, Fausto Giunchiglia, Anastasios Kementsietsidis, John Mylopoulos, Luciano Serafini and Ilya Zaihrayeu. *Data Management for Peer-to-Peer Computing : A Vision*. In WebDB, pages 89–94, 2002. 8

[Bhattacharjee & Goel 2005] Rajat Bhattacharjee and Ashish Goel. *Avoiding ballot stuffing in eBay-like reputation systems*. In Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems, P2PECON '05, pages 133–137, New York, NY, USA, 2005. ACM. 48

[Bianchini *et al.* 2005] Monica Bianchini, Marco Gori and Franco Scarselli. *Inside PageRank*. ACM Trans. Internet Technol., vol. 5, pages 92–128, February 2005. 48

[Bisnik & Abouzeid 2007] Nabhendra Bisnik and Alhussein A. Abouzeid. *Optimizing random walk search algorithms in P2P networks*. Comput. Netw., vol. 51, pages 1499–1514, April 2007. 45

[bit 2001] *Bittorrent Protocol Specification*. http://wiki.theory.org/BitTorrentSpecification, 2001. Retrieved 6th March 2011. 9

[Bloom 1970] Burton H. Bloom. *Space/time trade-offs in hash coding with allowable errors*. Commun. ACM, vol. 13, pages 422–426, July 1970. 39, 46, 91, 145

[Books 2010] LLC Books. Personality tests: Myers-briggs type indicator, purity test, minnesota multiphasic personality inventory, oxford capacity analysis. General Books LLC, 2010. 184

[Borisov 2006] Nikita Borisov. *Computational Puzzles as Sybil Defenses*. In 6th IEEE International Conference on Peer-to-Peer Computing, 2006 (P2P 2006), pages 171–176, September 2006. 51, 167

[Broder *et al.* 2000] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins and Janet Wiener. *Graph structure in the Web.* In Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking, pages 309–320, Amsterdam, The Netherlands, The Netherlands, 2000. North-Holland Publishing Co. 26, 28

[Buford *et al.* 2009] J.F.K. Buford, H.H. Yu and E.K. Lua. P2p networking and applications. Morgan Kaufmann series in networking. Elsevier/Morgan Kaufmann, 2009. 5

[cac 2005] *Cache Logic Study of P2P traffic.* http://www.slyck.com/story914_CacheLogic_Study_P2P_is_C 2005. Retrieved 6th March 2011. 1

[Camarillo & IAB 2009] G. Camarillo and IAB. *Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability.* RFC 5694 (Informational), November 2009. 4

[Castro *et al.* 2002a] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron and Dan S. Wallach. *Secure routing for structured peer-to-peer overlay networks.* SIGOPS Oper. Syst. Rev., vol. 36, pages 299–314, December 2002. 9, 33, 53

[Castro *et al.* 2002b] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec and Antony Rowstron. *SCRIBE: A large-scale and decentralized application-level multicast infrastructure.* IEEE Journal on Selected Areas in Communications (JSAC, vol. 20, page 2002, 2002. 7

[Chamberlin & Robie 2007] SimÂt'eon J. Boag S. Florescu D. FernÂt'andez M.F. Chamberlin D. and J. Robie. *XQuery 1.0: An XML query language.* http://www.w3.org/TR/2007/REC-xquery-20070123/, 2007. Retrieved 6th March 2011. 19

[Chawathe *et al.* 2003] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham and Scott Shenker. *Making gnutella-like P2P systems scalable.* In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM '03, pages 407–418, New York, NY, USA, 2003. ACM. 45

[Chen & Hwang 1992] Shu-Jen J. Chen and C. L. Hwang. Fuzzy multiple attribute decision making: Methods and applications. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992. 147

[Chen & Nahrstedt 1998] Shigang Chen and K. Nahrstedt. *On finding multi-constrained paths.* In Communications, 1998. ICC 98. Conference Record.1998 IEEE International Conference on, volume 2, pages 874 –879 vol.2, jun 1998. 129

[Cheng & Friedman 2005] Alice Cheng and Eric Friedman. *Sybilproof reputation mechanisms*. In Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems, P2PECON '05, pages 128–132, New York, NY, USA, 2005. ACM. 48

[Cheng & Friedman 2006] Alice Cheng and Eric Friedman. *Manipulability of PageRank under Sybil Strategies*. In First Workshop on the Economics of Networked Systems (NetEcon06), 2006. 48

[Chu *et al.* 2002] Jacky Chu, Kevin Labonte and Brian Neil Levine. *Availability and Locality Measurements of Peer-to-Peer File Systems*. In Proc. ITCom: Scalability and Traffic Control in IP Networks II Conference, volume SPIE 4868, pages 310–321, July 2002. 75, 101, 132, 150

[Clarke *et al.* 2001] Ian Clarke, Oskar Sandberg, Brandon Wiley and Theodore W. Hong. *Freenet: A Distributed Anonymous Information Storage and Retrieval System*. In INTERNATIONAL WORKSHOP ON DESIGNING PRIVACY ENHANCING TECHNOLOGIES: DESIGN ISSUES IN ANONYMITY AND UNOBSERVABILITY, pages 46–66. Springer-Verlag New York, Inc., 2001. 16

[Condie *et al.* 2004] T. Condie, S.D. Kamvar and H. Garcia-Molina. *Adaptive peer-to-peer topologies*. In Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on, pages 53 – 62, aug. 2004. 43

[Cormen *et al.* 2009] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. Introduction to algorithms. The MIT Press, 3rd édition, 2009. 131

[Cox *et al.* 2002] On P. Cox, Christopher D. Murray and Brian D. Noble. *Pastiche: making backup cheap and easy*. In In OSDI: Symposium on Operating Systems Design and Implementation, pages 285–298, 2002. 9, 48

[Crespo & Garcia-Molina 2002] Arturo Crespo and Hector Garcia-Molina. *Routing Indices For Peer-to-Peer Systems*. In Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02), ICDCS '02, pages 23–, Washington, DC, USA, 2002. IEEE Computer Society. 46, 89

[Danezis & Mittal 2009] George Danezis and Prateek Mittal. *SybilInfer: Detecting Sybil Nodes using Social Networks*. In NDSS, February 2009. 54

[Danezis *et al.* 2005] George Danezis, Chris Lesniewski-laas, M. Frans Kaashoek and Ross Anderson. *Sybil-resistant DHT routing*. In In ESORICS, pages 305–318. Springer, 2005. 53

[Daswani *et al.* 2002] Neil Daswani, Hector Garcia-Molina and Beverly Yang. *Open Problems in Data-Sharing Peer-to-Peer Systems*. In Proceedings of the 9th International Conference on Database Theory, ICDT '03, pages 1–15, London, UK, 2002. Springer-Verlag. 18, 39, 122

[Depken & Gregorius 2008] Craig A. Depken and Brandon Gregorius. *Auction Characteristics, Seller Reputation, and Closing Prices: Evidence from eBay Sales of the iPhone*. SSRN eLibrary, 2008. 48

[Devine 1993] Robert Devine. *Design and Implementation of DDH: A Distributed Dynamic Hashing Algorithm*. In 4th International Conference on Foundations of Data Organization and Algorithms (FODO, pages 101–114, 1993. 15, 19

[Dinger & Hartenstein 2006] J. Dinger and H. Hartenstein. *Defending the Sybil attack in P2P networks: taxonomy, challenges, and a proposal for self-registration*. In Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on, page 8 pp., april 2006. 47, 52

[Dingledine *et al.* 2004] Roger Dingledine, Nick Mathewson and Paul Syverson. *Tor: The Second-Generation Onion Router*. In In Proceedings of the 13th USENIX Security Symposium, pages 303–320, 2004. 48

[Douceur 2002] John R. Douceur. *The Sybil Attack*. In Revised Papers from the First International Workshop on Peer-to-Peer Systems, IPTPS '01, pages 251–260, London, UK, 2002. Springer-Verlag. 33, 34, 36, 47, 50, 51

[edo 2000] *eDonkey2000 Protocol Specification*. http://hydranode.com/docs/ed2k/ed2kproto.php, 2000. Retrieved 6th March 2011. 1

[Edwards 2002] J Edwards. Peer-to-peer programming on groove. Addison-Wesley, Indianapolis,, 2002. 8

[Ester *et al.* 1996] Martin Ester, Hans-Peter Kriegel, Joerg Sander and Xiaowei Xu. *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. In Evangelos Simoudis, Jiawei Han and Usama M. Fayyad, editeurs, Second International Conference on Knowledge Discovery and Data Mining, pages 226–231. AAAI Press, 1996. 192

[Fan *et al.* 2000] Li Fan, Pei Cao, Jussara Almeida and Andrei Z. Broder. *Summary cache: a scalable wide-area web cache sharing protocol*. IEEE/ACM Trans. Netw., vol. 8, pages 281–293, June 2000. 92, 101

[Ferguson & Huston 1998] P. Ferguson and G. Huston. Quality of service: delivering qos on the internet and in corporate networks. Wiley, 1998. 120, 128

[Fisk 2005] A. A Fisk. *Gnutella Dynamic Query Protocol v0.1*. http://www.ic.unicamp.br/ celio/peer2peer/gnutella-related/gnutella-dynamic-protocol.htm, 2005. Retrieved 28th March 2011 (last modified on 1 Apr 2005). 45

[Foster & Kesselman 2004] I. Foster and C. Kesselman. The grid: blueprint for a new computing infrastructure. The Morgan Kaufmann Series in Computer Architecture and Design Series. Elsevier, 2004. 5

[Foster *et al.* 2001] Ian Foster, Carl Kesselman and Steven Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. Int. J. High Perform. Comput. Appl., vol. 15, pages 200–222, August 2001. 4

[Fowlkes & Mallows 1983] E. B. Fowlkes and C. L. Mallows. *A Method for Comparing Two Hierarchical Clusterings*. Journal of the American Statistical Association, vol. 78, no. 383, pages 553–569, 1983. 193

[Ge & Cai 2008] Ping Ge and Hailong Cai. *Providing differentiated QoS for peer-to-peer file sharing systems*. Operating Systems Review, vol. 42, no. 6, pages 17–23, 2008. 121

[gnu 2000] *Gnutella Protocol Specification Version 0.4*. http://rfc-gnutella.sourceforge.net/developer/stable/index.html, 2000. Retrieved 6th March 2011. 1, 40

[gnu 2002] *Gnutella Protocol Specification Version 0.6*. http://rfc-gnutella.sourceforge.net/src/rfcdraft.html, 2002. Retrieved 6th March 2011. 91

[Gu *et al.* 2007] Peng Gu, Jim Wang and Hailong Cai. *ASAP: An Advertisement-based Search Algorithm for Unstructured Peer-to-peer Systems*. In Parallel Processing, 2007. ICPP 2007. International Conference on, page 8, sept. 2007. 46

[Gummadi *et al.* 2003] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy and John Zahorjan. *Measurement, modeling, and analysis of a peer-to-peer file-sharing workload*. SIGOPS Oper. Syst. Rev., vol. 37, pages 314–329, October 2003. 90

[Gyongyi & Garcia-Molina 2005] Zoltan Gyongyi and Hector Garcia-Molina. *Web Spam Taxonomy*. In First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb 2005), April 2005. 48

[Halkidi *et al.* 2001] Maria Halkidi, Yannis Batistakis and Michalis Vazirgiannis. *On Clustering Validation Techniques*. Journal of Intelligent Information Systems, vol. 17, pages 107–145, 2001. 193

[Hu *et al.* 2002] Yih-Chun Hu, Adrian Perrig and David B. Johnson. *Ariadne: a secure on-demand routing protocol for ad hoc networks*. In Proceedings of the 8th annual international conference on Mobile computing and networking, MobiCom '02, pages 12–23, New York, NY, USA, 2002. ACM. 48

[hua Chu *et al.* 2002] Yang hua Chu, S.G. Rao, S. Seshan and Hui Zhang. *A case for end system multicast*. Selected Areas in Communications, IEEE Journal on, vol. 20, no. 8, pages 1456 – 1471, oct 2002. 9

[Huebsch 2008] Ryan Jay Huebsch. *PIER: Internet Scale P2P Query Processing with Distributed Hash Tables*. PhD thesis, EECS Department, University of California, Berkeley, May 2008. 8

[ipo 2010] *Internet Study 2008/2009*. http://ipoque.com/sites/default/files/ mediafiles/documents/internet-study-2008-2009.pdf, 2010. Retrieved 6th March 2011. 1, 2

[Iyer *et al.* 2002] Sitaram Iyer, Antony Rowstron and Peter Druschel. *Squirrel: a decentralized peer-to-peer web cache*. In Proceedings of the twenty-first annual symposium on Principles of distributed computing, PODC '02, pages 213–222, New York, NY, USA, 2002. ACM. 8

[Jaccard 1901] Paul Jaccard. *Étude comparative de la distribution florale dans une portion des Alpes et des Jura*. Bulletin del la Société Vaudoise des Sciences Naturelles, vol. 37, pages 547–579, 1901. 193

[Jaffe 1984] Jeffrey M. Jaffe. *Algorithms for finding paths with multiple constraints*. Networks, vol. 14, no. 1, pages 95–116, 1984. 129

[Jain & Dovrolis 2003] Manish Jain and Constantinos Dovrolis. *End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput*. IEEE/ACM Trans. Netw., vol. 11, pages 537–549, August 2003. 143

[Janakiraman *et al.* 2003] R. Janakiraman, M. Waldvogel and Qi Zhang. *Indra: a peer-to-peer approach to network intrusion detection and prevention*. In Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on, pages 226 – 231, june 2003. 8

[Jiang & Jin 2005] H. Jiang and S. Jin. *Exploiting dynamic querying like flooding techniques in unstructured peer-to-peer networks*. In Network Protocols, 2005. ICNP 2005. 13th IEEE International Conference on, page 10 pp., nov. 2005. 45

[Johnson & Dynes 2007] M. Eric Johnson and Scott Dynes. *Inadvertent disclosure - information leaks in the extended enterprise*. In In Workshop on the Economics of Information Security (WEIS, pages 7–8, 2007. 31

[Johnson *et al.* 2009] M. Eric Johnson, Dan McGuire and Nicholas D. Willey. *Why file sharing networks are dangerous?* Commun. ACM, vol. 52, pages 134–138, February 2009. 31

[Johnson 2007] G. Johnson. *Arrest in case of ID theft by file sharing*. USA Today, September 2007. 31

[joo 2007] *Joost*. http://en.wikipedia.org/wiki/Joost, 2007. Retrieved on 6th March 2011. 9

[Jyothi & Dharanipragada 2009] B.S. Jyothi and J. Dharanipragada. *SyMon: Defending large structured P2P systems against Sybil attack*. In Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on, pages 21 –30, sept. 2009. 54

[Kalogeraki *et al.* 2002] Vana Kalogeraki, Dimitrios Gunopulos and D. Zeinalipour-Yazti. *A local search mechanism for peer-to-peer networks*. In Proceedings of the eleventh international conference on Information and knowledge management, CIKM '02, pages 300–307, New York, NY, USA, 2002. ACM. 44, 60

[Kamvar *et al.* 2003] Sepandar D. Kamvar, Mario T. Schlosser and Hector Garcia-Molina. *The Eigentrust algorithm for reputation management in P2P networks*. In Proceedings of the 12th international conference on World Wide Web, WWW '03, pages 640–651, New York, NY, USA, 2003. ACM. 48

[Karger *et al.* 1997] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine and Daniel Lewin. *Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web*. In Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM. 30, 169, 172

[Kay & Robie 2007] FernÂt'andez M.F. Boag S. Chamberlin D. Berglund A. SimÂt'eon J. Kay M. and J. Robie. *XML path language (XPath) 2.0*. http://www.w3.org/TR/2007/REC-xpath20-20070123/, 2007. Retrieved 6th March 2011. 19

[kaz 2001] *Kazaa*. http://en.wikipedia.org/wiki/Kazaa, 2001. Retrieved 6th March 2011. 1, 6, 91

[Kendall & Smith 1939] M. G. Kendall and Babington B. Smith. *The Problem of m Rankings*. The Annals of Mathematical Statistics, vol. 10, no. 3, pages 275–287, September 1939. 197

[Keromytis *et al.* 2002] Angelos D. Keromytis, Vishal Misra and Dan Rubenstein. *SOS: secure overlay services*. In Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM '02, pages 61–72, New York, NY, USA, 2002. ACM. 8

[kon 2000] *Kontiki*. http://en.wikipedia.org/wiki/Kontiki, 2000. Retrieved on 6th March 2011. 8

[Kosiur 1998] D.R. Kosiur. Ip multicasting: the complete guide to interactive corporate networks. Wiley, 1998. 120, 122

[Kubiatowicz *et al.* 2000] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishan Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells and Ben Zhao. *OceanStore: an architecture for global-scale persistent storage*. SIGPLAN Not., vol. 35, pages 190–201, November 2000. 9

[Kumar *et al.* 2006] P. Kumar, S. Gopalan and V. Sridhar. *Multi-Attribute Based Similar Content Indexing in Hybrid Peer-to-Peer Networks*. In Networking and Services, 2006. ICNS '06. International conference on, page 24, july 2006. 46

[Kwok & Yang 2004] Sai Ho Kwok and Christopher C. Yang. *Searching the peer-to-peer networks: The community and their queries*. JASIST, vol. 55, no. 9, pages 783–793, 2004. 126, 127

[Leach *et al.* 2005] Paul J. Leach, Michael Mealling and Richard Salz. *A Universally Unique IDentifier (UUID) URN Namespace*. Internet RFC 4122, July 2005. 22

[Leontiadis *et al.* 2006] Elias Leontiadis, Vassilios V. Dimakopoulos and Evaggelia Pitoura. *E.: Creating and Maintaining Replicas in Unstructured Peer-to-Peer Systems*. Rapport technique, In 12th International Euro-Par Conference on Parallel Processing, 2006. 43

[Lesniewski-Laas & Kaashoek 2010] Chris Lesniewski-Laas and M. Frans Kaashoek. *Whanau: a sybil-proof distributed hash table*. In Proceedings of the 7th USENIX conference on Networked systems design and implementation, NSDI'10, pages 8–8, Berkeley, CA, USA, 2010. USENIX Association. 54

[Levine & Margolin 2006] Shields C. Levine B. N. and B.N. Margolin. *A Survey of Solutions to the Sybil Attack*. Rapport technique, University of Massachusetts Amherst, Amherst, MA, October 2006. 51

[Li & Garcia-Luna-Aceves 2006] Zhenjiang Li and J. J. Garcia-Luna-Aceves. *A distributed approach for multi-constrained path selection and routing optimization*. In Proceedings of the 3rd international conference on Quality of service in heterogeneous wired/wireless networks, QShine '06, New York, NY, USA, 2006. ACM. 129

[Li 2008] Jin Li. *On peer-to-peer (P2P) content delivery*. PeertoPeer Networking and Applications, vol. 1, no. 1, pages 45–63, 2008. 123

[Liang *et al.* 2005] Jin Liang, Steven Y. Ko, Indranil Gupta and Klara Nahrstedt. *MON: on-demand overlays for distributed system management*. In Proceedings of the 2nd conference on Real, Large Distributed Systems - Volume 2, WORLDS'05, pages 13–18, Berkeley, CA, USA, 2005. USENIX Association. 9

[Lin & Wang 2003] Tsungnan Lin and Hsinping Wang. *Search Performance Analysis in Peer-to-Peer Networks*. In Proceedings of the 3rd International Conference on Peer-to-Peer Computing, P2P '03, pages 204–, Washington, DC, USA, 2003. IEEE Computer Society. 59

[Litwin *et al.* 1993] Witold Litwin, Marie-Anne Neimat and Donovan A. Schneider. *LH: Linear Hashing for distributed files*. In Proceedings of the 1993 ACM SIGMOD international conference on Management of data, SIGMOD '93, pages 327–336, New York, NY, USA, 1993. ACM. 15

[Litwin *et al.* 1996] Witold Litwin, Marie-Anna Neimat and Donovan A. Schneider. *LH* - a scalable, distributed data structure*. ACM Trans. Database Syst., vol. 21, pages 480–525, December 1996. 15

[Lüscher & Scott 1971] M. Lüscher and I.A. Scott. The lüscher color test. Pocket Books. Pocket Books, 1971. 185, 186, 190

[Lv *et al.* 2002a] Qin Lv, Pei Cao, Edith Cohen, Kai Li and Scott Shenker. *Search and replication in unstructured peer-to-peer networks*. In Proceedings of the 16th international conference on Supercomputing, ICS '02, pages 84–95, New York, NY, USA, 2002. ACM. 40, 41, 43, 45

[Lv *et al.* 2002b] Qin Lv, Sylvia Ratnasamy and Scott Shenker. *Can Heterogeneity Make Gnutella Scalable?* In IPTPS, pages 94–103, 2002. 43

[M. 2000] Jovanovic M. *Modelling large-scale peer-to-peer networks and a case study of gnutella*. Master's thesis, Department of Electrical and Computer Engineering and Computer Science, University of Cincinnati, June 2000. 25

[Mamdani 1977] E H Mamdani. *Applications of fuzzy set theory to control systems: a survey*. Fuzzy Automata and Decision Processes, pages 1–13, 1977. 68

[Margolin & Levine 2007] N. Boris Margolin and Brian Neil Levine. *Informant: Detecting Sybils Using Incentives.* In Sven Dietrich and Rachna Dhamija, editeurs, Financial Cryptography, volume 4886 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2007. 55

[Marzolla *et al.* 2007] Moreno Marzolla, Matteo Mordacchini and Salvatore Orlando. *Peer-to-peer systems for discovering resources in a dynamic grid*. Parallel Comput., vol. 33, pages 339–358, May 2007. 5

[Medina *et al.* 2000] Alberto Medina, Ibrahim Matta and John Byers. *On the origin of power laws in Internet topologies*. Rapport technique, Boston University Computer Science Department, 2000. 27

[Meng *et al.* 2006] Shicong Meng, Cong Shi, Dingyi Han, Xing Zhu and Yong Yu. *A Statistical Study of Today's Gnutella*. In APWeb, pages 189–200, 2006. 61, 62

[Mieghem *et al.* 2001] Piet Van Mieghem, Hans De Neve and Fernando Kuipers. *Hop-by-hop quality of service routing*. Computer Networks, vol. 37, no. 3-4, pages 407 – 423, 2001. 129

[Myers & Myers 1995] Isabel Briggs Myers and Peter B. Myers. Gifts differing: understanding personality type. Davies-Black Pub., 1995. 184

[Naoumov & Ross 2006] Naoum Naoumov and Keith Ross. *Exploiting P2P systems for DDoS attacks*. In Proceedings of the 1st international conference on Scalable information systems, InfoScale '06, New York, NY, USA, 2006. ACM. 33

[nap 1999] *Napster*. http://en.wikipedia.org/wiki/Napster, 1999. Retrieved on 6th March 2011. 1, 3, 39

[Nejdl *et al.* 2002] Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér and Tore Risch. *EDUTELLA: a P2P networking infrastructure based on RDF*. In Proceedings of the 11th international conference on World Wide Web, WWW '02, pages 604–615, New York, NY, USA, 2002. ACM. 13

[Neve & Mieghem 2000] Hans De Neve and Piet Van Mieghem. *TAMCRA: a tunable accuracy multiple constraints routing algorithm*. Computer Communications, pages 667–679, 2000. 129, 161

[Newman 2001] M. E. J. Newman. *The structure of scientific collaboration networks*. Proceedings of the National Academy of Sciences of the United States of America, vol. 98, no. 2, pages 409–415, January 2001. 26

[Oram 2001] Andy Oram. Peer-to-Peer : Harnessing the Power of Disruptive Technologies. O'Reilly, March 2001. 3, 21, 26

[p2p 2008] *P2P-Next Generation Consortium*. http://www.p2p-next.org/, 2008. Retrieved on 6th March 2011. 1

[Pandurangan 2001] G. Pandurangan. *Building Low-Diameter P2P Networks*. In Proceedings of the 42nd IEEE symposium on Foundations of Computer Science, FOCS '01, pages 492–, Washington, DC, USA, 2001. IEEE Computer Society. 43

[Pearman & Albritton 1997] R.R. Pearman and S.C. Albritton. I am not crazy, i am just not you:. Davies-Black Pub., 1997. 184

[Plaxton *et al.* 1997] C. Greg Plaxton, Rajmohan Rajaraman and Andréa 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, SPAA '97, pages 311–320, New York, NY, USA, 1997. ACM. 15

[PPS 2010] *PPStream*. http://en.wikipedia.org/wiki/PPS.tv, 2010. Retrieved on 6th March 2011. 9

[PPT 2005] *PPTV*. http://en.wikipedia.org/wiki/PPTV, 2005. Retrieved on 6th March 2011. 9

[pro 2003] *"How Much Information?"*. http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/index.htm, 2003. accessed on 6th March 2011. 37

[Prud'hommeaux & Seaborne 2007] Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF (Working Draft)*. Rapport technique, W3C, March 2007. 20

[Raju & Kumar 2010] Srinivasa Raju and Nagesh Kumar. Multicriterion analysis in engineering and management. Prentice-Hall Of India Pvt. Ltd., 2010. 131, 140

[Rand 1971] William M. Rand. *Objective Criteria for the Evaluation of Clustering Methods*. Journal of the American Statistical Association, vol. 66, no. 336, pages 846–850, 1971. 193

[Ratnasamy *et al.* 2002] Sylvia Ratnasamy, Mark Handley, Richard M. Karp and Scott Shenker. *Topologically-Aware Overlay Construction and Server Selection*. In INFOCOM, 2002. 43

[Resnick *et al.* 2000] Paul Resnick, Ko Kuwabara, Richard Zeckhauser and Eric Friedman. *Reputation systems*. Commun. ACM, vol. 43, pages 45–48, December 2000. 48

[Rhea & Kubiatowicz 2002] S. C. Rhea and J. Kubiatowicz. *Probabilistic Location and Routing*. In INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, volume 3, pages 1248–1257, 2002. 46, 90, 101, 145

[Ripeanu 2001a] M. Ripeanu. *[15] Peer-to-Peer Architecture Case Study: Gnutella Network*. In Proceedings of the First International Conference on Peer-to-Peer Computing, P2P '01, pages 99–, Washington, DC, USA, 2001. IEEE Computer Society. 27, 28

[Ripeanu 2001b] M. Ripeanu. *Peer-to-peer architecture case study: Gnutella network*. In Peer-to-Peer Computing, 2001. Proceedings. First International Conference on, pages 99 –100, aug 2001. 27

[Rowaihy *et al.* 2007] H. Rowaihy, W. Enck, P. McDaniel and T. La Porta. *Limiting Sybil Attacks in Structured P2P Networks*. In INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE, pages 2596 –2600, may 2007. 51

[Rowstron & Druschel 2001] Antony Rowstron and Peter Druschel. *Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility*. In Proceedings of the eighteenth ACM symposium on Operating systems principles, SOSP '01, pages 188–201, New York, NY, USA, 2001. ACM. 9

[Saaty 1994] T L Saaty. *How to Make a Decision: The Analytic Hierarchy Process*. Interfaces, vol. 24, no. 6, pages 19–43, 1994. 133, 140, 142

[Saint-Andre 2005] Peter Saint-Andre. *Streaming XML with Jabber/XMPP*. IEEE Internet Computing, vol. 9, pages 82–89, September 2005. 8

[sam 2006] *SAMCRA (Self-Adaptive Multiple Constraints Routing Algorithm)*. http://totem.run.montefiore.ulg.ac.be/algos/samcra.html, 2006. Retrieved on 6th March 2011. 161

[Sariou *et al.*] Stefan Sariou, Krishna and Steven. *SProbe: A Fast Technique for Measuring Bottleneck Bandwidth in Uncooperative Environments*. In IEEE Infocomm 2002. 143

[Saroiu *et al.* 2002a] Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble and Henry M. Levy. *An analysis of Internet content delivery systems*. SIGOPS Oper. Syst. Rev., vol. 36, pages 315–327, December 2002. 1, 90, 123, 124, 125, 143

[Saroiu *et al.* 2002b] Stefan Saroiu, Krishna P. Gummadi and Steven D. Gribble. *A Measurement Study of Peer-to-Peer File Sharing Systems*. January 2002. 91, 92

[Sen & Wang 2002] S. Sen and J. Wang. *Analyzing peer-to-peer traffic across large networks*. In Internet Measurement Workshop, pages 137–150, Marseille, France, November 2002. 27

[set 1999] *SETI@home project*. http://setiathome.berkeley.edu/, 1999. Retrieved 6th March 2011. 3, 7

[sky 2003] *Skype*. http://en.wikipedia.org/wiki/Skype, 2003. Retrieved on 6th March 2011. 8

[Sobrinho 2003] João Luis Sobrinho. *Network routing with path vector protocols: theory and applications*. In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM '03, pages 49–60, New York, NY, USA, 2003. ACM. 129

[sop 2004] *SopCast*. http://www.sopcast.com/, 2004. Retrieved on 6th March 2011. 9

[Spearman 1987] C. Spearman. *The proof and measurement of association between two things. By C. Spearman, 1904.* The American journal of psychology, vol. 100, no. 3-4, pages 441–471, 1987. 197

[sta 2005] *StartSurfing Encyclopedia. "Petabyte"*. http://startsurfing.com/encyclopedia//p/e/t/Petabyte. 2005. Retrieved on 6th March 2011. 37

[Stoica *et al.* 2001] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek and Hari Balakrishnan. *Chord: A scalable peer-to-peer lookup service for internet applications*. In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM. 30, 168

[Stoica *et al.* 2002] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker and Sonesh Surana. *Internet Indirection Infrastructure*. In In Proceedings of ACM SIGCOMM, pages 73–86, 2002. 8

[Stutzbach & Rejaie 2005] D. Stutzbach and R. Rejaie. *Capturing accurate snapshots of the Gnutella network*. In IEEE Global Internet Symposium, pages 127–132, 2005. 28

[Stutzbach & Rejaie 2006] Daniel Stutzbach and Reza Rejaie. *Capturing Accurate Snapshots of the Gnutella Network*. In INFOCOM. IEEE, 2006. 28

[Sugeno 1985] Michio Sugeno. Industrial applications of fuzzy control. Elsevier Science Inc., New York, NY, USA, 1985. 70

[Tang & Dwarkadas 2004a] Chunqiang Tang and Sandhya Dwarkadas. *Hybrid global-local indexing for effcient peer-to-peer information retrieval*. In Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association. 90

[Tang & Dwarkadas 2004b] Chunqiang Tang and Sandhya Dwarkadas. *Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval*. In NSDI, pages 211–224, 2004. 46

[Tang *et al.* 2003] Chunqiang Tang, Zhichen Xu and Mallik Mahalingam. *pSearch: information retrieval in structured overlays*. Computer Communication Review, vol. 33, no. 1, pages 89–94, 2003. 90

[Thampi & Sekaran 2009] S.M. Thampi and C.K. Sekaran. *An Enhanced Search Technique for Managing Partial Coverage and Free Riding in P2P Networks*. International Journal of Recent Trends in Engineering, vol. 2, no. 1–6, pages 33–41, 2009. 45

[Tsoumakos & Roussopoulos 2003] Dimitrios Tsoumakos and Nick Roussopoulos. *Adaptive Probabilistic Search for Peer-to-Peer Networks*. In Proceedings of the 3rd International Conference on Peer-to-Peer Computing, P2P '03, pages 102–, Washington, DC, USA, 2003. IEEE Computer Society. 44

[Waldman *et al.* 2000] Marc Waldman, Aviel D. Rubin and Lorrie Faith Cranor. *Publius: A robust, tamper-evident, censorship-resistant, web publishing system*. In In Proc. 9th USENIX Security Symposium, pages 59–72, 2000. 13

[Wallach 2003] Dan S. Wallach. *A survey of peer-to-peer security issues*. In Proceedings of the 2002 Mext-NSF-JSPS international conference on Software security: theories and systems, ISSS'02, pages 42–57, Berlin, Heidelberg, 2003. Springer-Verlag. 33

[Wang & Crowcroft 1996] Zheng Wang and J. Crowcroft. *Quality-of-service routing for supporting multimedia applications*. Selected Areas in Communications, IEEE Journal on, vol. 14, no. 7, pages 1228 –1234, sep 1996. 128

[Wang & Wang 2006] Yufeng Wang and Wendong Wang. *On studying P2P topology based on modified fuzzy adaptive resonance theory*. In Proceedings of the 2006 international conference on Intelligent computing: Part II, ICIC'06, pages 410–420, Berlin, Heidelberg, 2006. Springer-Verlag. 43

[Wang *et al.* 2005] Honghao Wang, Yingwu Zhu and Yiming Hu. *An Efficient and Secure Peer-to-Peer Overlay Network*. In Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary, LCN '05, pages 764–771, Washington, DC, USA, 2005. IEEE Computer Society. 52, 167

[Watts & Strogatz 1998] Duncan J. Watts and Steven H. Strogatz. *Collective dynamics of 'small-world' networks*. Nature, vol. 393, no. 6684, pages 440–442, 1998. 26

[Xiao *et al.* 2005] L. Xiao, Yunhao Liu and L.M. Ni. *Improving unstructured peer-to-peer systems by adaptive connection establishment*. Computers, IEEE Transactions on, vol. 54, no. 9, pages 1091 – 1103, sept. 2005. 43

[Xie & Yang 2007] H. Xie and Y. Yang. *A Measurement-based Study of the Skype Peer-to-Peer VoIP Performance*. In The Sixth International Workshop on Peer-to-Peer Systems, February 2007. 8

[Yang & Garcia-Molina 2002] B. Yang and H. Garcia-Molina. *Improving search in peer-to-peer networks*. In Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on, pages 5 – 14, 2002. 40, 44, 45, 60, 61

[Yu *et al.* 2006] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons and Abraham Flaxman. *SybilGuard: defending against sybil attacks via social networks*. In Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications, SIGCOMM '06, pages 267–278, New York, NY, USA, 2006. ACM. 54

[Yuan & Yin 2007] Liu J. Yuan F. and C. Yin. *A Scalable Search Algorithm for Unstructured Peer-to-Peer Networks*. In Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing âĂŞ SNPD, pages 199–204, 2007. 44

[Yuan 2002] Xin Yuan. *Heuristic algorithms for multiconstrained quality-of-service routing*. IEEE/ACM Trans. Netw., vol. 10, pages 244–256, April 2002. 129

[Yurkewych *et al.* 2005] Matthew Yurkewych, Brian N. Levine and Arnold L. Rosenberg. *On the cost-ineffectiveness of redundancy in commercial P2P computing*. In Proceedings of the 12th ACM conference on Computer and communications security, CCS '05, pages 280–288, New York, NY, USA, 2005. ACM. 48

[Zadeh 1965] L.A. Zadeh. *Fuzzy Sets*. Information Control, vol. 8, pages 338–353, 1965. 61

[zat 2006] *Zattoo*. http://en.wikipedia.org/wiki/Zattoo, 2006. Retrieved on 6th March 2011. 8

[Zegura *et al.* 1996] Ellen W. Zegura, Kenneth L. Calvert and Samrat Bhattacharjee. *How to Model an Internetwork*. In INFOCOM, pages 594–602, 1996. 75, 98, 149

[Zeleny 1982] M. Zeleny. Multiple criteria decision making. McGraw-Hill, New York, 1982. 147

[Zeng *et al.* 2006] Wenjun Zeng, Heather Yu and Ching-Yung Lin. Multimedia security technologies for digital rights management, chapitre Introduction: Digital Rights Management. Academic Press, Inc., Orlando, FL, USA, 2006. 33

[Zhang & Hu 2007] Rongmei Zhang and Y.C. Hu. *Assisted Peer-to-Peer Search with Partial Indexing*. Parallel and Distributed Systems, IEEE Transactions on, vol. 18, no. 8, pages 1146 –1158, aug. 2007. 46, 89, 90

[Zhang *et al.* 2007] Haoxiang Zhang, Lin Zhang, Xiuming Shan and V.O.K. Li. *Probabilistic Search in P2P Networks with High Node Degree Variation*. In Communications, 2007. ICC '07. IEEE International Conference on, pages 1710 –1715, june 2007. 44

[Zhang *et al.* 2009] Ying Zhang, Houkuan Huang, Dong Yang, Hongke Zhang, Han-Chieh Chao and Yueh-Min Huang. *Bring QoS to P2P-based semantic service discovery for the Universal Network*. Personal Ubiquitous Comput., vol. 13, pages 471–477, October 2009. 121

[Zhao *et al.* 2006] Shanyu Zhao, Daniel Stutzbach and Reza Rejaie. *Characterizing files in the modern gnutella network: A measurement study*. In In Proceedings of SPIE/ACM Multimedia Computing and Networking, 2006. 61

[Zhou *et al.* 2003] Feng Zhou, Li Zhuang, Ben Y. Zhao, Ling Huang, Anthony D. Joseph and John Kubiatowicz. *Approximate object location and spam filtering on peer-to-peer systems*. In Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware, Middleware '03, pages 1–20, New York, NY, USA, 2003. Springer-Verlag New York, Inc. 8

[Zhu & Hu 2006] Yingwu Zhu and Yiming Hu. *Enhancing Search Performance on Gnutella-Like P2P Systems*. Parallel and Distributed Systems, IEEE Transactions on, vol. 17, no. 12, pages 1482 – 1495, dec. 2006. 43

[Zhuge *et al.* 2005] Hai Zhuge, Xue Chen and Xiaoping Sun. *Preferential walk: towards efficient and scalable search in unstructured peer-to-peer networks*. In Special interest tracks and posters of the 14th international conference on World Wide Web, WWW '05, pages 882–883, New York, NY, USA, 2005. ACM. 44

Publications

Conference Papers

1. K. Haribabu, Dayakar Reddy, Chittaranjan Hota, Antti Yla-Jaaski, and Sasu Tarkoma, Adaptive Lookup for Unstructured Peer-to-Peer Overlays, in Proc. of Third International Conference on Communication Systems & Middleware (IAMCOM/COMSWARE 2008), Bangalore, 5-10 January 2008, pp. 776-782.
2. K. Haribabu, Chittaranjan Hota, and Antti Yla-Jaaski, Indexing Through Querying in Unstructured Peer-to-Peer Overlay Networks, in Proc. of 11th Asia-Pacific Network Operations and Management Symposium (APNOMS 2008), Beijing, 22-24 October 2008. pp. 102-111, Springer, 2008.
3. K Haribabu, Chittaranjan Hota, Saravana S, Detecting Sybils in Peer-to-Peer File Replication Systems, in Proc. International Conference on Information Security and Digital Forensics, ISDF 2009, City University, London, Sept 2009, pp. 152-164, Springer, 2009.
4. Sirish Kumar, K Haribabu, Chittaranjan Hota, Efficient Search in Peer-to-Peer Networks Using Fuzzy Logic, in Proc. International Conference on Distributed Computing and Information Technology , ICDCIT 2010, Bhubaneshwar, Feb 2010, pp. 188-193, Springer, 2010.
5. K Haribabu, Dushyant Arora, Bhavik Kothari, Chittaranjan Hota, Detecting Sybils in Peer-to-Peer Overlays using Neural Networks and CAPTCHAs, in Proc. IEEE International Conference on Computational Intelligence and Communication Networks, CICN 2010, Bhopal, Nov 2010, pp. 154-161. IEEE Computer Society 2010.
6. K Haribabu, Arindam Paul, Chittaranjan Hota, Detecting Sybils in Peer-to-Peer Overlays using Psychometric Analysis Methods, in Proc. 25th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA-2011), Singapore, Mar 2011, pp. 114-119. IEEE Computer Society 2011.

Journal Papers

1. K Haribabu, Chittaranjan Hota, Securing Peer-To-Peer Overlays From Multiple Identity Forgery Attacks, International Journal of Advanced Engineering & Applications, Vol II, pp. 132-138, Jun 2010.
2. GAUR: A method to detect Sybil groups in Peer-to-Peer overlays, Haribabu K, Chittaranjan Hota, and A Paul, Int. J. Grid and Utility Computing, IJGUC, Vol. 3, Nos. 2/3, Inderscience, pp. 145-156, 2012.

3. K Haribabu, Chittaranjan Hota, A QoS Model for P2P Filesharing Networks (To be communicated).

4. K Haribabu, Chittaranjan Hota, Sirishkumar Balaga, A Novel Metric for Neighbour Selection in Query Forwarding (To be communicated).

5. K Haribabu, Chittaranjan Hota, A Novel Indexing Technique for Efficient Search in P2P (To be communicated).

Biographies

Brief Biography of the Candidate

K Haribabu is currently Lecturer in the Department of Computer Science at Birla Institute of Technology & Science, Pilani, Rajasthan, India. He has completed MSc.(Tech) in 2003 and Masters of Engineering degree in Software Systems in 2005. He was executive committee member of BITS Alumni Association for several years. He is a nucleus member of Academic Registration and Counselling Division and carried out several computerization tasks. He acted as a Web-Co chair for two reputed conferences. He is a member of IEEE. He presented papers at reputed international conferences COMSWARE'08, ICDCIT'10 and AINA'11. He was a reviewer for Student Symposium ICDCIT 2012. He teaches courses like Network Programming, Computer Programming, Database Systems, Object Oriented Programming etc. His research interests are in the areas of Peer-to-peer overlays, Mobile communication, and Quality of Service.

Brief Biography of the Supervisor

Chittaranjan Hota is currently Associate Professor in the Department of Computer Science and Information Systems at Birla Institute of Technology and Science, Pilani Hyderabad Campus, Hyderabad, India. He has earned his Bachelors degree in Computer Engineering, Masters degree in CSE, and PhD degree in CSE from Amravati (MS), TIET (deemed) and BITS, Pilani (deemed) in the year 1990, 1998, and 2006. He shoulders the responsibility of Head of the Computer Sc & Information Systems department from the inception of BITS Pilani Hyderabad campus (2008). He is also the Faculty In-Charge of Information Processing and Business Intelligence Division at BITS Hyderabad. He has worked in Academic and Research institutes within and outside India over past 22 years. Over these years he had been teaching and researching in areas of Computer Networks, Distributed Systems, Information Security, Network Programming, and Operating Systems. He has worked with teaching and research appointments at universities abroad like, School of Computer Science and Engineering, University of New South Wales, Sydney, Australia; Helsinki Institute of Information Technology, Helsinki, Finland; School of Mathematics & Engineering, City University, London; and Department of Electronics and Computer Engineering, University of Cagliari, Italy over past several years. He has published extensively in national and international conferences and journals. He is a life member of ISTE, India; member of IE, India and member of ACM. His research interests are in the areas of Traffic Engineering in IP Networks, Security and Quality of Service issues over the Internet, Peer-to-Peer Overlays, Mobile Wireless Networks, and Cloud Computing.