# Semantic Query Routing in Agent-Based P2P Systems

**Salman Elahi**


**Master of Science**
**(Informatics)**
**School of Informatics**
**University of Edinburgh**
**2005**

# Abstract

This thesis describes the development and analysis of distributed resource discovery techniques for use in an agent-based P2P information retrieval system. In a decentralised distributed environment where no central authority is present, a critical task is to identify the most suitable agent in the network. Agents have to cooperate to forward queries among themselves so as to find appropriate agents, return and merge results in order to fulfil an information retrieval task in a distributed environment. In our approach we have exploited social metaphors learned from social networks. Agents are connected through a '*knows'* relationships maintained by each agent. Initially agents have knows relationship with agents assigned to them at random. This initial relationship is known as a random topology. However, we show that such an approach can be significantly enhanced by memorising network communication in a lazy learning style. This memorisation constitutes *expertise-based knowledge* at each of the agent involved in the query. Agents use this topical knowledge to answer subsequent queries and improve their resource discovery performance over time. It also helps agents to evolve a semantic topology from a random one. This semantic topology creates semantic clusters of agents sharing similar expertise in the network. Experimental results indicate that our approach improves the performance of an agent-based P2P system with respect to criteria such as recall, precision and number of messages over time.

# Acknowledgements

All thanks are due to Allah the most merciful and most generous, who has created this entire universe, always guides us and gives us the strength that we need in order to achieve what we strive for in our lives.

I would like to thank our beloved prophet Hazrat Muhammad (P.B.U.H), whose teachings show us the correct path to follow in our lives.

I would like to gratefully acknowledge the enthusiastic supervision of my project supervisor, Dr. Dave Robertson, who has been a great source of inspiration for me throughout my project. He helped me and guided me in every possible way in order to make this project successful. I would never have been able to achieve what I have without his help and guidance. Whenever I got stuck in some problem, he gave me helpful suggestions and ideas to resolve that problem. His creative thinking and breadth of knowledge inspired me a lot and kept me motivated for striving for more and better.

I am forever indebted to my parents and sisters for being so supporting and heartening that I feel very fortunate to have such a family.

I would like to express my gratitude and appreciation to Christoph Tempich for his continuous support, guidance and encouragement.

Finally I would also like to thank my friends Zeeshan Pervez, Aitezaz Ali, Danish Najam, and Zahid Khan for their moral support, understanding, encouragement and endless patience when it was most required.

Thanks to all of you!

# Declaration

I declare that the thesis was composed by me, that the work contained herein is my own except where explicitly stated otherwise in the text, and this work has not been submitted for any other degree or professional qualification except as specified.

(Salman Elahi)

*Dedicated to Maan Gee & Abu Gee*

***Whatever I am today, I owe it to my beloved parents***

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This chapter gives an overview of the thesis by describing the problem, proposed solution, objectives and organisation of this report.

## 1.1. Problem Statement

During the last decade, with the exponential growth of the Internet the number of information sources available to the users has become extremely large. Though it has given a handful of resources to search through, it has also stemmed problems in the effectiveness of the retrieved information as current technologies are not scaling well with the growing size of the Web. Although it had been anticipated by many active and vociferous Internet researchers that the growth of the Internet is effectively unbounded. But unfortunately, this vision could not prove its effectiveness at least in one important area that is of searching the World Wide Web.

Individual Web servers and Web search engines are two distinct ways to search the Web. The former approach is practical when one is sure about the server which could contain the required information and that server should also provide search facility. Otherwise the later approach, Web search engines, is used. These search engines maintain centralized indices of the Web by brute force traversal and indexing each page found. Google[1], Altavista[2], and Yahoo[3] are the big names which provide the facility to search the Web by maintaining large amount of indices of Web pages spread all across the Internet. These search engines are the backbone of the Web but statistics (see below) show there is a very significant drop in the performance of the search engines as compared to the growth of Internet and the World Wide Web.

1 http://www.google.com
2 http://www.altavista.com
3 http://www.yahoo.com

Recent studies show that there are approximately 600 billion static pages present on the Web while Google, which claims to have indexed the largest amount of the Web pages, claims to maintain indices for approximately 3.083 billion web-pages. The Internet is being populated by 10,000,000 static pages each day while the search engines are growing by almost 10% of this pace [2]. Another downside along with not being able to cope with the pace of web growth is growing latency in indexing records, with a typical wait of several months for new pages to be indexed. This is quite evident from the number of links pointing to non-existent documents in search results which is almost 25% [2]. One of the main reasons of this is the inability to perform "full" Web traversal in a reasonable time. This situation clearly depicts that existing solutions are not scaling well.

## 1.2. Proposed Solution

However, over the last few years, introduction of the Peer-to-Peer (hereafter P2P) systems have revolutionised the way of information exchange and information retrieval in distributed systems. P2P systems are decentralized distributed systems consisting of logically distinct computing elements called *peers*. These peers have comparable roles and responsibilities. They share or consume services and resources amongst each other [3]. The concept of P2P systems is backed up by the increasing power of PCs in terms of speed and processing power since late 1990s. This power enabled many server side software to be deployed on individual PCs. These machines now can easily handle a direct two-way flow of information between peers on the Web. This paradigm is particularly well suited for distributed resource sharing and information searching. As search procedures can be conducted on different peers or locations [4]. It can latently cover content-related/appropriate resources in reasonable time which is the biggest problem faced by the Internet search i.e. maintenance of such huge indices etc.

The advent of the Semantic web [16] has enlivened the concept of intelligent agents. These agents can use intelligent reasoning to search the web which would be containing ontologically marked up (machine readable metadata) content instead of simple content present in the current web [16]. The need to attach metadata with resources on the Web also indicates that current techniques are not scaling well to meet the up coming

challenges. It gives the idea of reducing the granularity of the Web from web-page level artefacts to entities more akin to the facts in knowledge base of an expert system.

Resource discovery is the task of searching distributed systems for objects with required characteristics. A solution which could address the problem of scalability of the resource discovery task depends upon the distributed nature of the system used to share the index-building load. In the case of the Web, this solution requires each Web server to maintain its own indices. P2P systems possess such functionalities but they suffer from a lack of mechanisms for intelligent query evaluation, intelligent query routing, learning, self organisation and autonomous decision making etc. The basic concepts underlying agent paradigm and P2P systems are similar. The synthesis of these two paradigms could be exploited to deal with the challenges being faced by the current Web. Agent paradigm can offer concepts and techniques to be used in application modelling and design level of P2P systems (i.e. ontologies to share network resources in a semantically rich manner, protocols for negotiation etc.). A P2P paradigm can offer a wide range of practical application domains, state of the art implementation techniques, and core infrastructure components [3].

To find the appropriate information in such distributed systems, a query must be evaluated (at least) on each peer. If peers pass a summary of their contents (expertise) along with the normal communication, next time a peer can locate and query only those peers which contain relevant answers, a technique known as query routing. This process can be divided into three distinct sub processes: *Peer Selection,* identifying the one who is most likely to contain the answer to the query; *Query Evaluation,* documents satisfying the query criteria are identified; and *Information Access,* documents are retrieved matching the identifiers. To exploit the query routing mechanism to its full extent, a peer must have appropriate knowledge about other peers, so that it is able to effectively guide the queries to their answering destinations.

Agent-based P2P computing is a suitable technique for query routing systems as its computation model combined with the distributed ad-hoc nature of the problem resembles the social interactions in a group of autonomous systems. In fact, resource discovery is quite similar to a *service discovery* task or a *connection* task in which agents in a multi-agent system tries to find other agents which can provide them the required

service [6]. Furthermore, self-organisation is the most desirable feature of such systems. It is the ability to dynamically adjust with the changes in the environment without any external support. Systems mimic self-organised behaviour by the execution of several individual components that interact locally to achieve an overall combined goal. The main characteristic of self-organised systems is that they achieve complex collective tasks with relatively simple individual behaviours in a decentralised environment [20].

## 1.3. Objectives

This thesis work is a part of a joint research study under taken by two MSc students Salman Elahi and Zeeshan Pervez under the supervision of Dr. Dave Robertson. This joint research study has various purposes:

- Development of an intelligent self-organising query routing mechanism.
- Evaluation of the synthetic effect of two paradigms; multi-agent systems and P2P systems in a simulated environment.
- Comparison of two different approaches to multi-agent systems:

    - Agents who are intelligent themselves and learn over time (software agents (see section 2.3 for details))
    - Agents with intelligent protocols which evolve over time (protocol-based agents (see section 7.2 for a summary))

This thesis is about intelligent agents while intelligent protocols have been developed and studied by Pervez [34] in the other part of this joint research study. This work is closely related to SWAP [17]. The basic idea of memorising network information has been picked from [7, 8] but it has been used with several modifications i.e. query routing, search, indexing mechanisms etc. to suit our simulation requirements.

## 1.4. Outline

This section describes the outline of this thesis report:

Chapter 2 describes basic concepts behind P2P and multi-agents systems, the possible performance gains which could be gained by their synthesis. It also describes the related work done so far in these directions.

Chapter 3 describes basic concepts and techniques, along with their alternates, used in the development of this system with examples. It gives the basic understanding of those techniques to the interested readers.

Chapter 4 describes simulator used and the user interface to give a visual feel of the whole system to the reader before proceeding to the technical details.

Chapter 5 describes the system overview with detailed explanations of system architecture and mechanisms used for communication, query routing, and searching.

Chapter 6 describes the evaluation criteria with detailed discussion of the results.

Chapter 7 describes the comparison of the two approaches for multi-agent systems, mentioned above. Intelligent agents versus intelligent protocols

Chapter 8 describes the conclusion of the evaluation and comparison of results.

Chapter 9 describes future work for this project and the related research interests.

# Chapter 2

# Background and Literature Review

This chapter describes the underlying concepts of P2P and agent-oriented computing. We will also look at the synthetic effect of these two paradigms along with some related literature review.

## 2.1. P2P Systems

There is no commonly agreed upon definition of P2P systems. The literature proposes a wide variety of definitions; however, the following are the most common [35]:

- P2P refers to a class of systems that offer efficient techniques for resource discovery and sharing in a decentralised manner.
- P2P computing offers a network-based solution for sharing resources and services via direct exchange.

In a typical client/server architecture, one or more computers are designated as servers depending upon the network size. The server, typically an unattended system, listens and responds to the requests of clients (other computers/systems in the network). In contrast to this dependent approach, in a P2P system every computer, referred to as *peer*, acts as a client with a layer of server functionality. This allows the peers to act as a client as well as a server sharing the network load and resources. P2P systems introduce an independence culture replacing dependency on central severs. Peers can perform their tasks independently. They can listen and respond to the requests of other peers in the network.

## 2.2. P2P Architectures

P2P systems are generally split into the following two categories with respect to their architecture [36, 37]:

- Centralised
- Semi-Centralised

### 2.2.1. Centralised Architecture

These systems that have no central control or authority over each peer and in which each is considered equally capable and independent are said to have a *pure* P2P architecture. They share and consume resources of the network. They can communicate directly with each other and are aware of each other constantly. The other way of peer communication is indirect communication where peers communicate through other intermediate peers. Indirect communication network could be of two types with respect to the organisation of peers:

***Structured networks:*** in this type of network, peers are organised with a regular structure. Each peer maintains information about subset of other peers in the network. This information is maintained as a distributed routing table typically referred to as Distributed Hash Tables (DHTs). These routing tables are used to provide efficient lookup and mapping between the resources (e.g. file) and location (e.g. node). CAN [40], Pastry [41], and Chord [39] are examples of structured systems.

***Unstructured networks:*** in this type of network, peers are not organised according to some structure. An indirect unstructured architecture no longer needs to enforce a specific network configuration; however, the focus and work load is shifted towards resource discovery as network can change without alerting the peers. Napster [43] and Gnutella [44] are examples of such systems.

### 2.2.2. Semi-Centralised Architecture

This type of P2P architecture is also known as hybrid architecture. In this architecture there is at least one central point of control. This control could be for many purposes e.g. from the implementation of strict control policies to simply provide a central indexing service to the other peers. The central peer is used for maintaining the central indices of the resources available in the network as well as providing processing capabilities. These networks do not need resource discovery services as the central peer maintains an index. SETI@home is an example of this type of system [42].

## 2.3. Agents: A Well Suited Paradigm for P2P

The basic underlying concepts of a peer and an agent are very similar and lead to the synthetic approach for these two paradigms which could reveal new direction for distributed computing. The driving force of this approach is based on the concept that the shortcomings of P2P systems (see section 2.4 for details) can be made up by the capabilities of agent paradigm which we will discuss in the later sections.According to the literature there is no agreed upon definition for what is an agent; however, there are certain numbers of characteristics which are commonly associated with agents by the research communities [45]:

- ***Autonomy:*** Agents can control their operations and internal states without the intervention of humans or outside computer programs.

- ***Reactivity:*** Agents react to the changes in the environment and adjust accordingly to achieve their goals.

- ***Social ability:*** Agents can interact with other agents or humans either to cooperate or compete. This ability lays the ground for multi-agent system.

- ***Pro-activity:*** As agents can react to their environment they can take initiatives to achieve their goals as well.

- ***Learning/Adaptation:*** Agents learn from their previous experiences and evolve their behaviour accordingly to improve the efficiency and effectiveness in carrying out their tasks.

- ***Mobility:*** Agents can move from one node to another node to in the network.

### 2.3.1. A Peer with Agent Capabilities

By adapting these characteristics (mentioned above), a peer can also enhance its performance.

- ***Autonomy:*** a peer having the autonomy can decide when to enter and when to leave the network, whether to answer or reject the query etc.
- ***Reactivity:*** a peer can react to the changes in the environment.
- ***Social ability:*** can introduce *social values* in P2P systems like *trust and reputation*.
- ***Pro-activity:*** can be useful to take initiative to achieve goals and to alert the users, based upon their interests, of any changes in the network.
- ***Adaptation and Mobility:*** can enable peers to implement efficient learning algorithms for resource discovery in order to improve performance.

## 2.4. Limitations of Peers

The following are some important limitations of peers with respect to agent capabilities:

- ***Communication:*** languages used, at present, for communication in P2P are very simple; supporting only well defined concepts rather than complex semantically rich communication.

- ***Data Model:*** peers exchange very simple data models e.g. files and directory structures only.

- ***Data Integration:*** peers operate in well defined environments and do not tackle the problems of heterogeneity and inconsistency which are inherent with distributed systems.

- ***Routing:*** intelligent routing techniques can play a critical role, especially in the absence of any centralised authority.

However, these limitations can be improved by the merger of these two paradigms. In this merger each peer would have an associated agent which employs an efficient communication language, can exchange complex data structures e.g. in business-to-business, ecommerce applications, can operate in heterogeneous environments, and has efficient and intelligent routing algorithms.

## 2.5. Trust and Reputation: An Issue for Agent-Based P2P Systems

In a P2P network, peers share and consume resources. When a peer issues a service request it does not know whether it will be granted or will not be granted, by virtue of autonomy of other peers in the network. Therefore, concepts of *trust and reputation* also are applicable to the social behaviour of peers [46, 5, 1]. The concept of trust is slightly different from that of password based security. Passwords reveal the identity of an agent but do not guarantee a service. The following are the major challenges to deal with:

- How an agent can decide when and where it should reveal its identity
- How an agent can trust other agents with whom it has no previous experience

In [1] a mechanism has been described to award prizes and penalties. According to this mechanism, when a service request is issued only those agents will respond which are reasonably confident that they can provide a suitable answer. This approach mimics social behaviour. Agents are ranked e.g. if they provide a wrong answer their trust level decreases in the community and vice versa. To discourage deceiving behaviour a good reputation is built slowly but drops rapidly.

## 2.6. Literature Review

In decentralised unstructured distributed systems, a critical task is to identify the node which is most likely to contain the answer to the query and pass that query to that node, considering the number of messages generated on the network for this routing. Our work relates to two research areas in different ways, we will discuss them in the following sections:

### 2.6.1. P2P Systems

In general the emphasis of the research in P2P systems has been on investigating new possibilities for efficient and effective network topologies or document distribution in the network.

In [22] an approach based on social network analysis has been discussed named as the small-world-effect. They have studied the topologies to establish efficient connections among peers in the network. Their approach exploits the power law link distribution of naturally occurring networks. In a power law distribution peers show a higher tendency to connect some particular peers as compared to other peers. In this distribution few peers will have high degree of connectivity and many peers will have low degree of connectivity. Their approach introduces a sophisticated change in basic Gnutella [44] approach (flooding the network). First, they route the query to those peers which have a high degree of connectivity instead of broadcasting the query to all known peers. Second, peers exchange their contents with two other known peers which have lower degree of connectivity in an ascending order. Peers also keep their known peers updated about their network position (degree of connectivity) by update

messages, in case a peer leaves or joins that peer. Experimental results show that queries can be answered in less number of hops as compared to basic Gnutella approach. However, the drawback of this approach is that it generates extra network traffic because of continuous exchange of contents and update messages between peers.

In contrast, our approach uses semantic similarity value to select a peer to which to route the query instead of degree of connectivity of a peer. This approach seems to work in highly replicated environment. For example, in an mp3 file sharing environment this approach has higher probability to find the required file by routing the query to a peer with a high degree of connectivity but in our case if one wants to search a Bibtex reference related to the hardware field then it is not reasonable to route the query to a software expert peer which has a high degree of connectivity.

Another approach has been discussed in [23] to identify the resources in least messages. Each peer holds a subset of a virtual distributed search tree like a distributed hash table. This tree defines the position of a peer and the content it is responsible for, in the network in the form of a binary string. The combinations of bits in the binary string are abstracts for different data items in the network. Queries are routed to the peers which have the most similar representative string in the tree. In contrast to our approach they do not propose any solution to find the most relevant resource in the first place. This direction of research has mostly been investigated by the other research community which we will discuss in the following section.

### 2.6.2. Agent-based P2P Systems

The use of schemas and ontologies in P2P systems has revolutionised this field. It has enabled P2P systems to take advantages and overcome their limitations by adapting recently developed semantic web technologies. This direction of research is focusing on synthetic effects and the benefits which could be achieved by the merger of these two paradigms.

In [28] a similar approach, named as ReferralWeb, with slight differences has been discussed. They also have taken inspiration from social networks to route the query. They analyse social networks by mining the public resources available on the Web. These resources could be of many types such as links to homepages, Internet directories, list of people, organisational charts etc. They have also used expertise-based model like us. A peer routes the query to the peer which is most likely to answer the query, based on the similarity value between peer expertise and query topic. Like our content holder indices they use referrals to other peers to route the query. However, they have not used counsellor and net-worker indices. Experimental results show that better precision could be achieved by using referrals as compared to the random selection. It also ensures that peers adapt dynamic topology and form group of peers sharing similar expertise. They contradict with our approach in the way of learning. We learn by observing and memorising the network communication while they learn and adapt the topology by mining the Web. They have claimed that the number of messages also decreases with their approach but no empirical evidence has been given.

In [27] an expertise based model has been discussed for efficient query routing. It is similar to our approach in a sense that each query has a 'focus' (topic in our case) associated with it, which is a part of an ontology. When a peer receives a query their proposed algorithm tires to match the focus of the query with the contents in the knowledge base of the peer. This matching is done in two ways. First, syntactic matching which is quite straightforward as the algorithm matches the keywords associated with the contents in the knowledge base. Second, semantic matching for which algorithm tries to match the context of the contents with the focus of the query. If it finds any relevant contents those are sent as answers to the querying peer otherwise query is routed to the peer which has similar contents to the focus of the query. In contrast to our approach they do not evolve their learning process by observing network communication.

Another semantic approach for query routing has been discussed in [9]. They learn the network by advertising the peer expertise in the network while we learn the network in a lazy learning style i.e. we observe the network communication and use it later for responding to new queries instead of advertisement. Like us they also use a flat P2P

network architecture i.e. every peer is equally capable and independent. However, peers are not considered equal in their knowledge about a particular field. They simply keep the replier as their known peers but we use complex mechanism to store and reuse this information later. Experimental results show that our approach performs well and adapts the semantic topology resulting in the form of semantic clusters of peers sharing common expertise.

EDUTELLA [24] is a semantic P2P system, its basic functionality is very similar to that of SWAP [25] project which uses an expertise based model for query routing. In [26] they have discussed a query routing mechanism. Peers with common expertise are arranged in hypercube topology which ensures that each peer is contacted exactly once for a query. This approach is contrary to ours, as we do not consider each peer equally capable of answering a query. We rank peers according to their knowledge about a particular field of expertise and also consider their previous performance in answering the queries related to that particular field. Our experiments show that we also need fewer queries to reach the most knowledgeable peer.

# Chapter 3

# Basic System Concepts

This chapter has been to provide the reader with basic underlying concepts of the techniques which have been used in this thesis.

## 3.1. Learned Paradigms

There are two distinct types of intelligent learning algorithms being practised in the field of AI:

- Lazy Learning
- Eager Learning

### 3.1.1. Lazy Learning

They have the following distinguished characteristics which draw a boundary line between them and the other types of algorithms [11, 12]:

Algorithms based on lazy learning approach defer the compilation of the input data until an explicit request for a prediction is received.
The input data could be used in a combined way to satisfy the incoming request more effectively and efficiently.

Case-based reasoning (CBR) and K-nearest neighbours are the most commonly used examples of lazy learning algorithms. We discuss the characteristics of CBR along with some exemplar scenarios. CBR uses facts and observations stored in the knowledge base not the rules encoded out of them. When a new problem comes; it is matched against the cases stored in the knowledge base and similar cases are retrieved to suggest a solution which is reused and tested for success. It is repeated until a

successful solution is found. On success the new solution and the problem encountered are stored in the knowledge base for future reference. Domains suitable for CBR solutions usually possess the characteristics such as [29]:

- Previous records of instances are kept and maintained carefully
- Acquired experiences are documented in repositories
- Experts explain tasks by examples

Following are exemplar scenarios where CBR is used:

- ***Diagnosis:*** Case-based systems are widely used in medical diagnosis domain. When a new patient comes; its symptoms are matched against the symptom lists of the previous cases. The retrieved cases are then used to suggest a diagnosis for the current case.
- ***Decision Support:*** In making complex decisions, CBR systems are very helpful. It is natural to look for similar problems to get a hint for the possible solution. CBR systems are particularly very helpful in querying structured, modular and heterogeneous repositories.

### 3.1.2. Eager Learning

On the other hand, as evident from the name, eager learning algorithms posses the following characteristics [11, 12]:

- They process the input data immediately which results in the form of some rule sets, decision trees, or neural networks etc. They store only the results (which normally are generalisations) and discard the data.
- They satisfy the incoming requests based upon this a priori induced information.

Decision trees, neural nets and Bayesian classifiers are the most commonly used examples of the eager learning algorithms. We discuss the characteristics of decision trees along with a real world problem. A Decision tree is a graph of decisions and

their possible outcomes. It takes the value of an independent attribute (variable, which at least has two or more than two values) and by following some rules (in the form of conditions) produces a value for the dependent attribute (variable, which at least has two or more than two values). Decision trees start with a *root node* having different *branches* leading to other *child nodes* or *leaves* (tips at the bottom of the graph) as shown in Figure 1. These leaves always represent decisions reached by following certain conditions. We demonstrate this by a standard example [14]:

Suppose, we schedule our weekend for five options depending upon certain social and environmental factors. The options could be 'No Outing', 'Play Football', 'BBQ', 'Theatre', 'Shopping' and the social and environmental factors could be Overtime (Yes, No), Weather (Sunny, Windy, Cloudy), Good Movie (Yes, No). We can plan considering different possibilities like; if there is overtime then we will not go for outing. If there is no overtime and it's sunny, then we will play football, but if it's windy, and there is a good movie, then we will go for theatre. If there is no overtime, it's windy and no good movie, then we will go to shopping. If there is no overtime and it's rainy, then we will go for BBQ.

To remember all this, we can draw a flowchart (as shown in Figure.1) for our convenience which will enable us to quickly reach on the decision. Such diagrams are called **Decision Trees.** After having this decision tree, scheduling our weekend will not be a hectic job. We simply have to follow the graph according to the facts and our weekend scheduled would be ready.

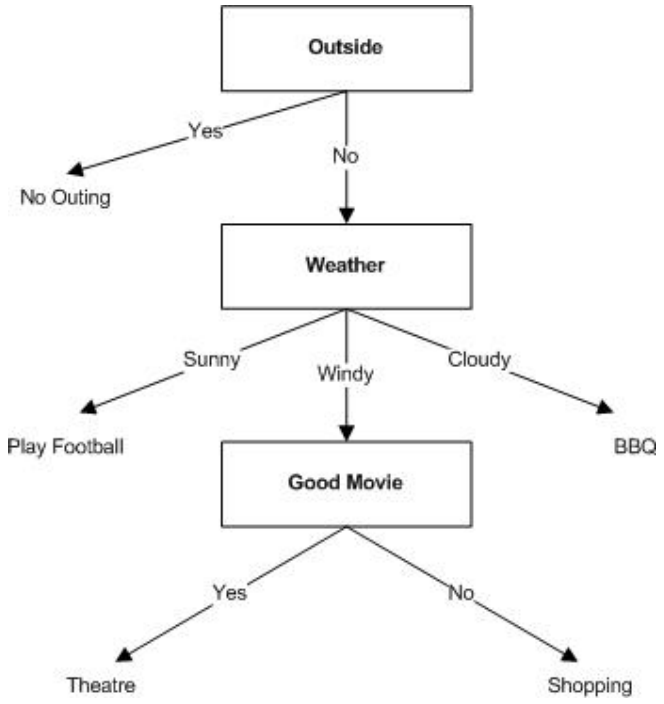



**Figure 1: A simple decision tree**

This decision tree can be drawn manually but there are automated techniques to build large and complex decision trees by learning from examples. However, we will not go into their details as that is out of scope of our thesis. Readers can find further information in [14].

### 3.1.3. Comparison of Lazy and Eager Learners

To summarise, lazy learners do not process the input data and use the original information to guide the decision making when needed. On the other hand, eager learners process the inputs immediately and use the processed results for decision making.

The most important characteristic of lazy learners is the flexibility of usage of stored information to answer queries. The basic assumption behind eager learner is that their learning bias is appropriate for the performance of the task. As far as this assumption is true, it can yield performance benefits. But it is risky because there is a significant

chance of loosing crucial information during the immediate processing of the input data which could play a vital role in generating accurate responses to the queries, if that assumption happens to be wrong. Thus, lazy learners in one sense are more trust worthy to handle unanticipated queries than eager learners.

The following are some heuristics on advantages and disadvantages of lazy learners over eager learners and vice versa:

- Lazy learners are well suited for incremental learning tasks because of their low learning costs.
- Lazy learners can be efficient problem solvers as they store and adapt solutions for later reuse.
- Lazy learners can generate detailed explanations instead of abstract explanations, which are preferable for many tasks.
- Lazy learners could be fine tuned by using cached information about prediction quality.
- Eager learners use fewer resources at the time of problem solving as they save pre compiled information while the lazy learners do the compilation at the time when problem comes.
- Eager learners need less storage capacity as they only store abstraction of the information in the form of rules.
- Eager learners may be more appropriate for time critical decision making tasks

## 3.2. ACM Topic Hierarchy

The Association for Computing Machinery [33] is the organization which maintains one of the biggest on-line digital libraries along with other major activities i.e. ACM Press, ACM Portal etc. ACM also maintains a topic hierarchy for almost all the computer science related topics known as the ACM topic hierarchy or more formally as the ACM Computing Classification System (CCS). The benefit of defining this hierarchy is to ensure accurate categorization of the documents to provide a quick content reference to the reader. This categorization also facilitates the on-line search in the ACM digital library or other on-line resources.

The ACM CCS consists of a four-level tree. Three of these levels are coded according to standard terminology while the fourth one is an uncoded level for subject description etc. The tree consists of eleven first levels nodes, the number of second and third level nodes varies with respect to each of the first level nodes. However, on average second level sparse to four nodes and to maximum it goes to eight nodes, whereas, third level consists of six to thirteen leaves. At fourth level these are just uncoded subject descriptions if any. Figure.2 shows a small fragment from the ACM topic hierarchy down to three levels starting from 0 for ACM root. Where dotted lines mean there could be more levels to come.

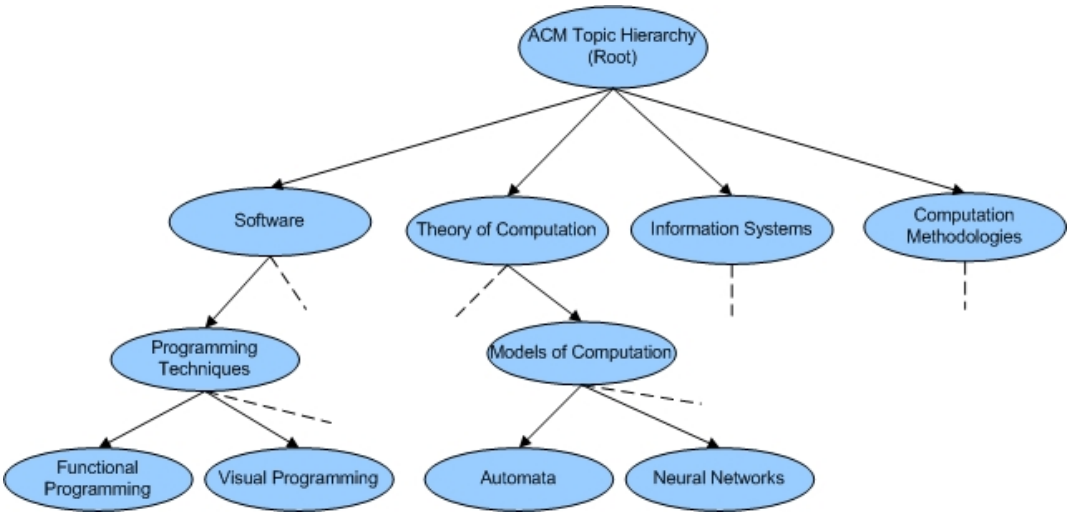



**Figure 2: A small fragment from the ACM topic hierarchy**

The ACM CCS covers almost 1287 different topics. First level nodes are labelled with capital alphabets from A to K and covers topics like General Literature, Hardware, Computer Systems Architecture to Information Systems and Computing Milieux etc. while second and third levels are labelled as a combination of alphabets and numeric [9, 10]. An author has to provide the categorization of her/his writing from the CCS. It can be explained with a brief example, suppose, there is a paper to be published about Information Storage and Retrieval which addresses the issues like content analysis, selection process, information networks etc. This paper has to provide the following categorization from CCS before submission to the ACM:

"*H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing -- Abstracting methods; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval -- Selection process; H.3.4 [Information Storage and Retrieval]: Systems and Software -- Information networks*"

This way it would be published, stored, referred, and reviewed under the selected categories which facilitate the user to access this paper later on.

## 3.3. Semantic Similarity Function

Finding semantic similarity between words or concepts has long been studied in Information Retrieval and Natural Language Processing. It has been a fundamental issue in a wide variety of applications of computational linguistics and Artificial Intelligence. The similarity between two words or concepts is measured by the similarity in concepts underlying the two different words or concepts. Two types of methods are used to measure similarity between concepts [8]:

### 3.3.1. Edge Counting-Based

An edge counting-based method is used in tree-like taxonomies, where nodes represent concepts (ACM topics in our case). The minimum number of edges separating two topics $t_1$ and $t_2$ is a metric for measuring the conceptual distance of t1 and $t_2$. These hierarchies consist of 'IsA' or 'Is a Kind of' relations. Such relations play an important role in determining the semantic similarity between two topics. Sometimes, taxonomies have irregular densities of links among concepts due to their broad domain. In that case, depth of the topic in the hierarchy with respect to the common subsumer, along with the minimum number of edges separating the concepts, is used. It is defined as in the following equation:

$$S(t_1, t_2) = \begin{cases} e^{-\alpha l} \cdot \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}} & \text{if } t_1 \neq t_2, \\ 1 & \text{otherwise} \end{cases}$$

where $l$ is the length of the shortest distance between the two topics $t_1$ and $t_2$ in the graph spanned by the *subTopic* relation (in our case) and depth $h$ is the level, in the tree, of the direct common subsumer of $t_1$ and $t_2$. Parameters $\alpha$ and $\beta$ are used to scale the contribution of shortest path $l$ and depth $h$, respectively. There values have been set to $\alpha = 0.2$ and $\beta = 0.6$ based upon the benchmark data set given in [8]. There might be some exceptions to these values but in our case they worked well. According to [15] it has been proven that the minimum number of edges separating two concepts $con_1$ and $con_{2\ is}$ a metric for measuring the conceptual difference. The reason to introduce $h$ in the calculation is that the concepts existing at higher levels in taxonomies are more general and semantically less similar than concepts at lower levels.

Let's demonstrate the use of this similarity function with an example: Suppose, we have to calculate semantic similarity among three topics to decide which pair is the most similar one. Say for instance, those topics are visual programming, functional programming, and neural networks. This is graphically represented in Figure.2.

By putting the values of $l$ and $h$ into the above equation for first pair, we get the similarity value, where $l = 6$ and $h = 1$:

*Sim ('visual programming', 'neural networks') = 0.15*

For the second pair, we get the similarity value, where $l = 2$ and $h = 3$:

*Sim ('visual programming', 'functional programming') = 0.63*

For the third pair, we get the similarity value, where $l = 6$ and $h = 1$:

*Sim ('functional programming', 'neural networks') = 0.15*

These values suggest that the second pair of topics is more similar than the first and third ones.

### 3.3.2. Information Theory-Based

An information theory-based method is another approach for calculating similarity between concepts in taxonomies. This approach is used in large information repositories to handle the problem of varying link distances encountered by edge-counting based approach. It calculates similarity between two concepts by capturing the maximum information content of the concept that commonly subsumes the two concepts [15]. The information content depends upon the probability of the occurrences of the concept in taxonomy. The probability of a concept is calculated upon the number of instances a concept and its sub concept occur in the taxonomy. The information content is then defined as negative the likelihood of the probability. It is defined as in the following equation [8]:

$$\text{Sim}(c_1, c2) = \max_{c \in S(c_1, c_2)} [-\log p(c)]$$

Here $S(c_1,c_2)$ is the set of concepts which subsumes both $c_1$ and $c_2$. For every c out of $S(c_1,c_2)$ a probability would be calculated with negative the likelihood and the maximum is chosen as the information content of the concept. We shall not demonstrate this here with an example as we are only dealing with tree like taxonomies in this system. However, interested readers can find further information in [15].

## 3.4. Peers/Agents

A peer is an individual, independent computing process. Our system architecture proposes that there is an intelligent agent associated with each peer and owns a complex knowledge base of its own. By combining a peer with an intelligent agent resources could be used more efficiently and effectively. In this way, peers in P2P systems become autonomous entities which could now initiate tasks, make intelligent decisions to pursue their goals, and have effective communication with other peers. So, from here onwards, peer and agent points to the same thing, preferably we will be using the word 'agent'.

We assume an agent has the following generic characteristics [13]:

- An agent is independent of other agents.
- An agent has intelligence to take initiatives and timely decisions.
- An agent is acquainted with some other agents.
- A query should not be propagated indefinitely between agents.
- An agent may choose not to respond or forward some queries depending upon its position in the network.

## 3.5. Query Routing Mechanism

Query routing is the process of identifying the agent which is most likely to contain the answer to a query and passing the query to that agent. There are three distinct types of query routing mechanisms, which we discuss in the following subsections:

### 3.5.1. Overt

This is the simplest form of query routing. In this mechanism every agent communicates directly with the querying agent (the one who receives the query from the user) without the involvement of any intermediate agents (who receives the query from the querying agent and passes to next known agents). When an agent receives a query from the user, it evaluates the query against its knowledge base and sends the query to its known agents (whom it knows in the network). These agents check the query against their local knowledge base and refer to their known agents, whom they found appropriate to answer the query, in the answer message to the querying agent. The querying agent then sends the query to those referred agents. This process keeps on until maximum hop count limit is reached (see section 4.2.6 for details).

This can be visualised by an example, suppose, there are three agents A, B, and C. A is querying agent and its known Agent is B. Agent B has E as its known agent. If A sends query to Agent B, it will check the query against its local knowledge base and in its answer message, to the querying agent, will refer to its known Agent E. Agent A

will then send query message to Agent E. Agent E will repeat the same process as Agent B if hop count allows otherwise simply will send answer message to Agent A. Figure.3 shows the overt routing mechanism where each agent has one known agent. A red line represents messages handled by the querying agent.

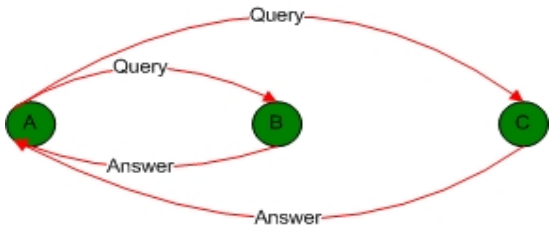



**Figure 3: Overt query routing mechanism**

### 3.5.2. Covert

Covert is a slightly modified form of the overt mechanism. In this approach, intermediate agents send queries directly to their known agents instead of referring them to the querying agent in the answer message. On receiving the answer from their known agents these intermediate agents send the answers to the querying agent. In this way, intermediate agents get directly involved in the routing process instead of indirect involvement (in case of overt). This process is repeated until a maximum hop count limit is reached (see section 4.2.6 for details).

An example communication using covert could be as follows. Suppose Agent A knows Agent B and Agent B knows Agent C. When Agent A receives a query from the user it sends it to its known Agent B. Agent B send answer message to Agent A if it found any and also forward query message to its known Agent C. Agent C send answer message to Agent B if it found any which is then sent to Agent A by Agent B. Similarly, Agent C can forward the query message to its known agents if maximum hop count is not reached yet. Figure.4 shows the covert routing mechanism where each agent has one known agent. A red line represents messages handled by the querying agent and blue lines represent messages handled by other agents.

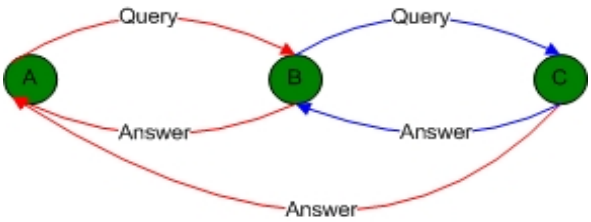



**Figure 4: Covert query routing mechanism**

### 3.5.3. Hybrid

Hybrid is a mixture of both overt and covert approaches. It forwards query messages covertly while answer messages are sent overtly. When a querying agent receives a query, it sends the query messages to its known agents. These known agents send answer messages to the querying agent and forward the query message to their known agents. These known agents now reply directly to the querying agent instead of replying to the intermediate agent. They can also forward the query message to their known agents if maximum hop count limit (see 4.2.6 for details) is not reached yet.

A hybrid approach is exemplified by the following scenario. Suppose Agent A knows Agent B and Agent B knows Agent C. When Agent A receives a query from the user it sends it to its known Agent B. Agent B will send answer message to Agent A if it found any and also forwards the query message to its known Agent C. Now Agent C will send the answer message directly to Agent A instead of sending to intermediate Agent B. It can also forward the query message to its known agents if the hop count is not reached yet. Figure.5 shows the hybrid routing mechanism where each agent has one known agent. A red line represents messages handled by the querying agent and a blue line represents messages handled by other agents.

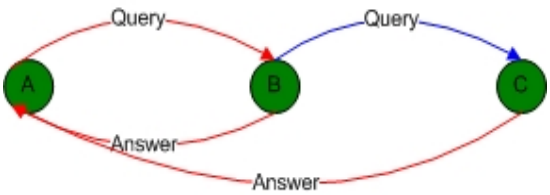



**Figure 5: Hybrid query routing mechanism**

### 3.5.4 Comparative Analysis

In this section, we compare the performance of overt, covert, and hybrid query routing mechanisms mentioned above. Table 1 presents the statistics in terms of number of messages generated using each of these query routing mechanisms. Figure 6, 7, 8 shows a communication scenario using these query routing mechanisms where each agent sends messages to two known agents. A red line represents messages handled by the querying agent and a blue line (if any) represents messages handled by other agents.

| ***Routing Mechanism*** | **$P_{Q=}1$** | | **$P_{Q=}2$** | | ***Increment Percentage*** | |
|---|---|---|---|---|---|---|
| | **Q** | **O** | **Q** | **O** | **Q** | **O** |
| **Overt** | 4 | 4 | 12 | 12 | 67% | 67% |
| **Covert** | 3 | 5 | 8 | 16 | 63% | 69% |
| **Hybrid** | 3 | 4 | 8 | 12 | 63% | 67% |

**Table 1: $P_Q$ represents the number of queried agents (agents whom an agent sends a query message). Q represents the number of messages handled by the querying agent and O represents the total number of messages generated over the network in one query session.**

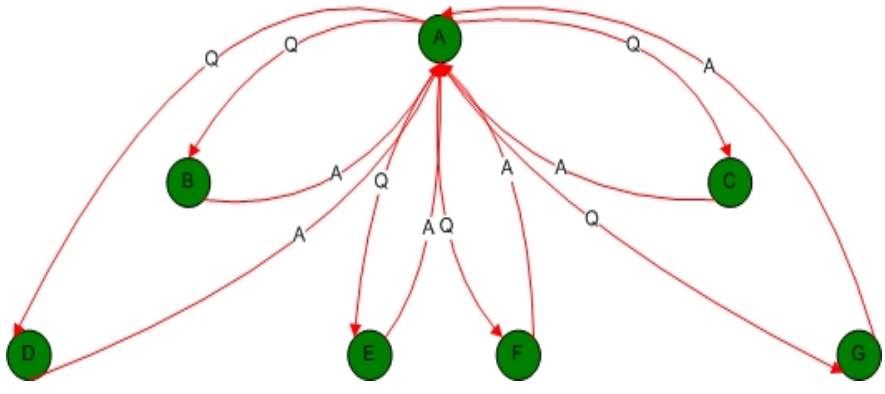



**Figure 6: Overt query routing mechanism, Q is for query message and A is for answer message**

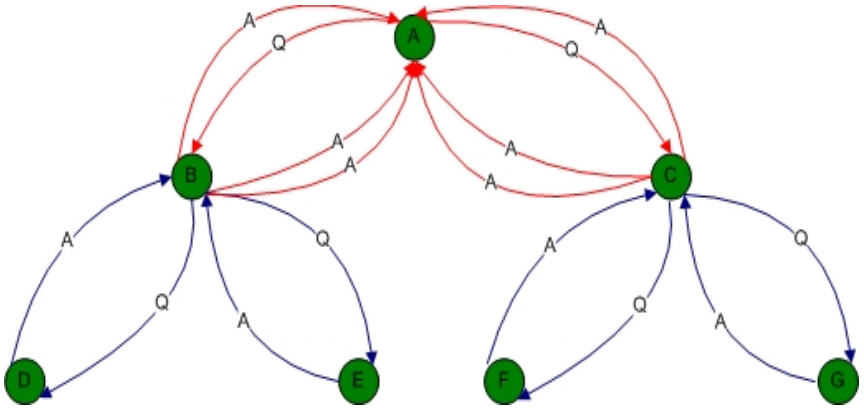



**Figure 7: Covert query routing mechanism, Q is for query message and A is for answer message**

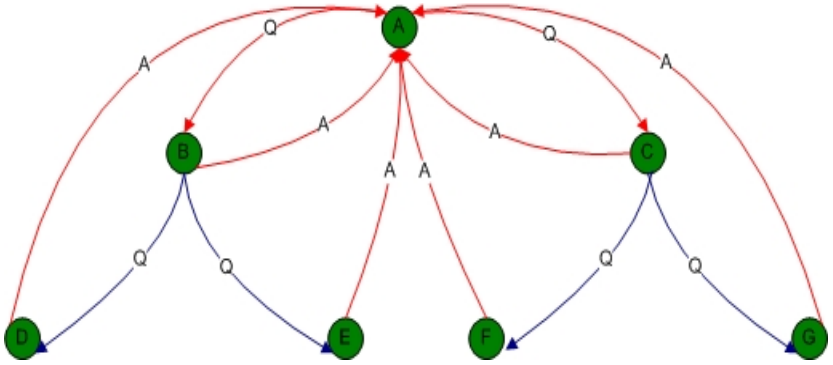



**Figure 8: Hybrid query routing mechanism, Q is for query message and A is for answer message**

From the statistics presented in the table 1 hybrid is the best query routing mechanism to follow. According to these statistics *overt* put all the communication load on the querying agent. Intermediate agents just have to communicate twice in a query session. Once for receiving the query message and secondly to answer the query while the querying agent participate in every communication on the network for a particular query session. As the network size and the number of queried agents increases as the communication load on the querying agent increases. One queried agent overt produces 4 messages while in case of two queried agents it produces 12 messages with an increment of 67%.

*Covert* is as good as *hybrid* in putting communication load on the querying agent, as is evident from the statistics in the table 1 but it generates more messages than hybrid in terms of total number of messages per query session. It also produces latency in answer messages as the repliers send answer messages to intermediate agents first and then intermediate agents forward them to the querying agent. Overt is efficient in this regard as the answer messages are sent directly to the querying agent. However, hybrid has best of both of these approaches, lesser communication load on the querying agent, and lesser overall messages with efficient communication mechanism.

Based upon these results, we have chosen *hybrid* as the query routing mechanism for our system.

# Chapter 4

# Simulator and User Interface Description

This chapter describes the details of the simulator used to develop this system, and the user interface used in this system. These details would be helpful to the reader to visualise the query routing process.

## 4.1. Simulator

We have used a network simulator developed by Dr. Jane Hilston and her PhD student Yusuf Abushaban (School of Informatics, University of Edinburgh) to simulate our agent-based P2P system. It is one of the main contributions of this project is to use this network simulator to simulate such networks as it has never before been used for this purpose (hitherto it was used to simulate simple distributed algorithms in distributed systems course). In P2P environment peers are independent of each other and process operations locally in parallel. To simulate such behaviours we required a simulator that can help us in executing operations simultaneously. Since this simulator is developed in Java which has built-in support for multi-threaded applications. We chose to use it to simulate our agent-based P2P system.

### 4.1.1. Basic Features

This simulator has several interesting features which enable a user to simulate a distributed network in a few mouse clicks. A user can create an agent by a left click which is displayed as a blue circle with its Id on it. Similarly more agents could be created. This network is saved in a XML file. This file contains the information about every agent created in the network regarding their Ids, colour, failed status, x/y-coordinates of the screen where they were drawn by the user etc. It allows the user to store the created network and use it any time later by loading it back.

First an agent is always selected by the user. For this purpose, user has to left click on an already created agent that turn its colour to yellow. Colour is changed to distinguish the first agent from other agents which have blue colour. A user can make a communication link between two agents by a right click on each of them. But this communication would be one way i.e. only first agent can send message to the second agent not vice versa. For two way communication a user has to repeat the same process from the second agent to the first agent. A user can also demonstrate fault tolerant behaviour by setting the failed status of an agent to true. This status is toggled by clicking the centre button of the mouse. Communication among agents is demonstrated by a tiny yellow leaflet showing the information about the message type (query, answer), sender Id, and receiver Id. The features discussed in this section are the features of the original simulator. However, we have introduced many changes to suite our requirements which we will discuss in the section below.

### 4.1.2. Changes and Additional Features to Suit Our System

Various changes and additional features have been introduced in this simulator to suit our system requirements especially in user interface. Initially the user interface for this simulator was very simple i.e. just a wide empty screen where user can create agents, communication links between them etc. to simulate distributed algorithms. However, to simulate our agent-based P2P system effectively we have embedded several new components into the user interface. Figure 9 shows the user interface for the original simulator, where a user has created four agents and communication links among them. Yellow colour represents the first agent which starts the communication and the black lines represent communication links.

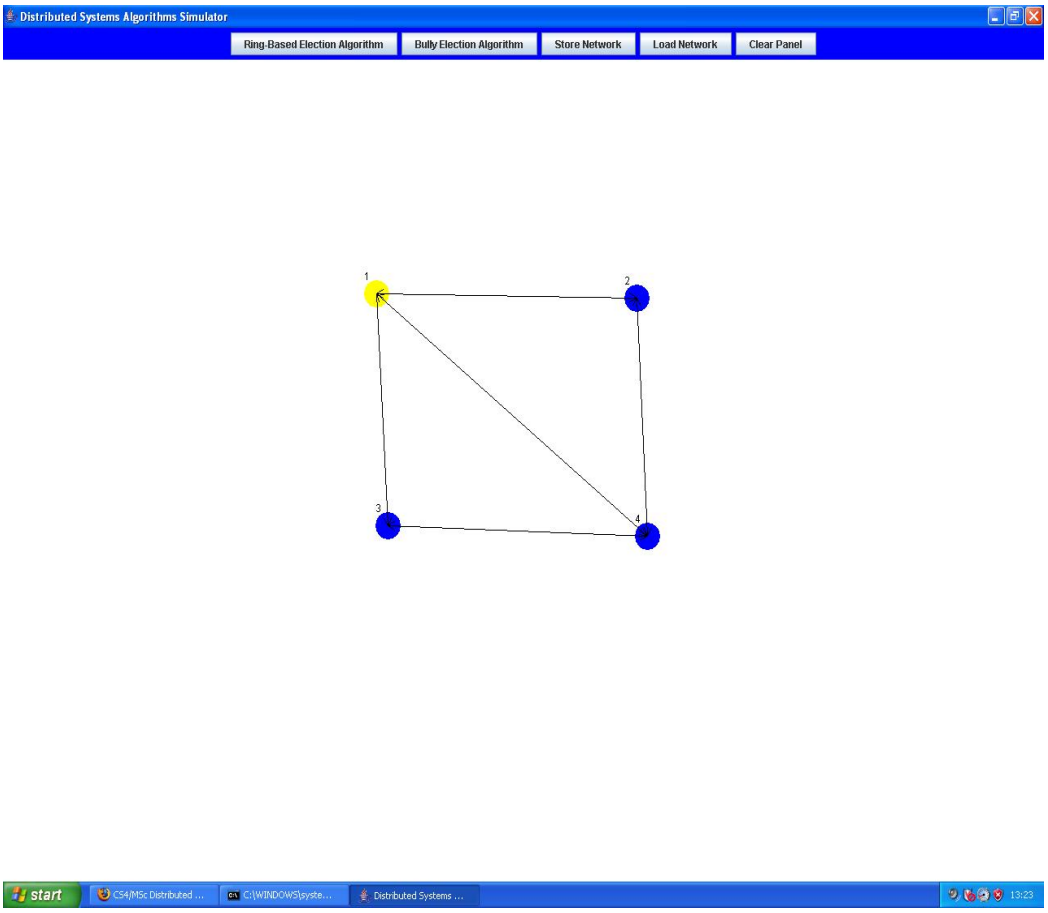



**Figure 9 : User interface for the original simulator**

### 4.1.3. New User Interface

To simulate our agent-based P2P system effectively, we have embedded the following components into the existing user interface:

***Agent Area (Central Panel)*:** To give a visual representation of agent communication we added a central panel named as 'agent area' (as shown in Figure 10). As we have the assumption that agents in our system will not leave or join the network during a query session so the user does not need to create and store networks. He/She can just load a predefined network. The user also does not need to create communication links as the agents know with which agents they can communicate (see section 5.2.11, 5.2.12 for details). This communication link is always a two way communication link. For example if Agent A can communicate with Agent B it is implicit that Agent B can

communicate with Agent A. We have tested our simulation for three different sizes of networks. Initially we simulated a network of 20 agents and then for scalability tests we upgraded it to a network of 40 and 60 agents respectively.

***Message Board (Right Panel):*** To show the agent communication and other agent activities in a log format which could be used later even after the query, we have added a right panel named as 'message board' which displays all available data on communication among agents in a particular query session. This board displays query reception messages, local knowledge base look up activity status, answer messages, query forwarding message and hop count etc (as show in Figure 14).

***ACM Topic Hierarchy Tree (Left Panel):*** We have also added the ACM topic hierarchy (see section 3.2 for details about ACM topic hierarchy) in a drop down tree format to enable the user to select the topic of the query which helps to select the first agent. As a user will no longer be able to select the first agent so there will be no yellow circles. User does not have to create communication links as well because agents are assigned random known agents (with whom an agent can communicate) initially which are changed as they learn the network for similar expertise. No fault tolerant behaviour is needed as we are dealing with semantic network issues rather physical network. So, we have the assumption that our network is fault free (see section 4.3.1 for details).

***Algorithm Choice List:*** We have also added a drop down list which gives a choice to select an algorithm out of three available algorithms (as shown in Figure 12). These three algorithms are for evaluation purposes, otherwise in an ideal case user will always select the third option (QUROMIDI with query relaxation and randomness) (see section 6.4.4 for details).

***Search Fields:*** We have also added three fields to enable the user to search a bibliographic reference by title, author, and year (as shown in Figure 13). Each agent in our simulation has 20 bibliographic references. There are 400 records in a network of 20 agents. For scalability tests we upgraded the size of knowledge base to 800 and 1200 records for a network of 40 and 60 agents respectively.

***Table of Retrieved Records:*** A table of records have been added to display the retrieved results. This table shows the record Id, title, authors, year of publication and its relevance with the keywords of the query entered by the user (as shown in Figure 14). This value could vary in the range of .50 to 1.0.

***Pop Up Window:*** A user can see complete details in a pop up window, in Bibtex format, of any of the retrieved records by just clicking on it (as shown in Figure 15). An example of Bibtex record is:

*@article{590909,*
*author = {Stefan Wermter},*
*title = {Knowledge Extraction from Transducer Neural Networks},*
*journal = {Applied Intelligence},*
*volume = {12},*
*number = {1-2},*
*year = {2000},*
*issn = {0924-669X},*
*pages = {27--42},*
*publisher = {Kluwer Academic Publishers},*
*address = {Hingham, MA, USA},*
*}*

## 4.2. A Stepwise Description of User Interaction

This section will enable the user to visually see the changes which have been discussed in the above section with a stepwise description of how to use the interface. The user interface consists of the following components:

- Central panel (Agent area)
- Right panel (Message board)
- Left panel (ACT tree)
- Algorithm choice list
- Search Fields
- Table of retrieved records
- Pop Window

When the user loads the system, the following interface is loaded at first, as shown in Figure 10:

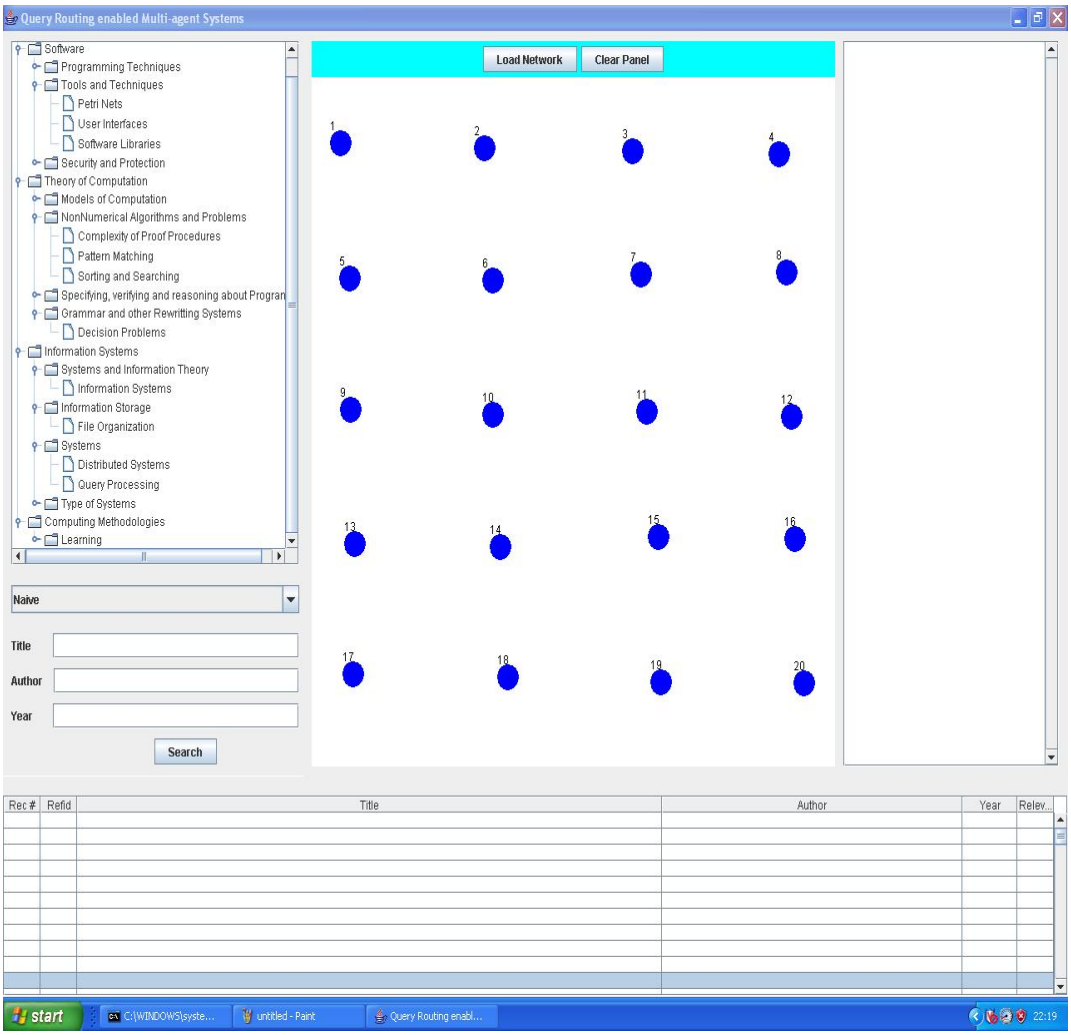



**Figure 10: User interface at start up**

To start a search for a bibliographic reference a user has to select a topic from the ACM tree, that is give in the left panel. This topic is used to select the first agent when user presses the search button after entering all necessary details for the query. Figure 11 shows topic selection:

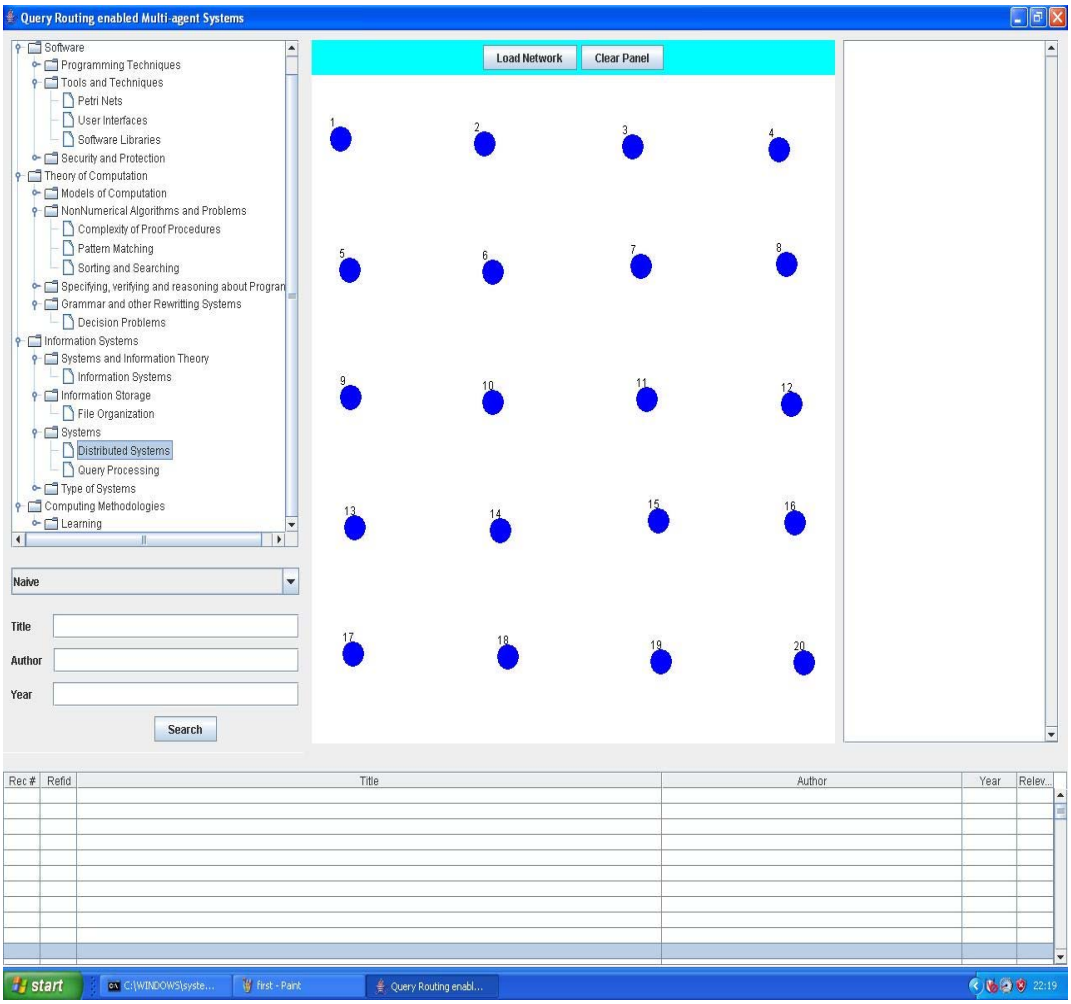



**Figure 11: Topic selection from ACM topic hierarchy**

Then user has to select a search algorithm from the drop down list, Figure 12 shows the algorithm selection:

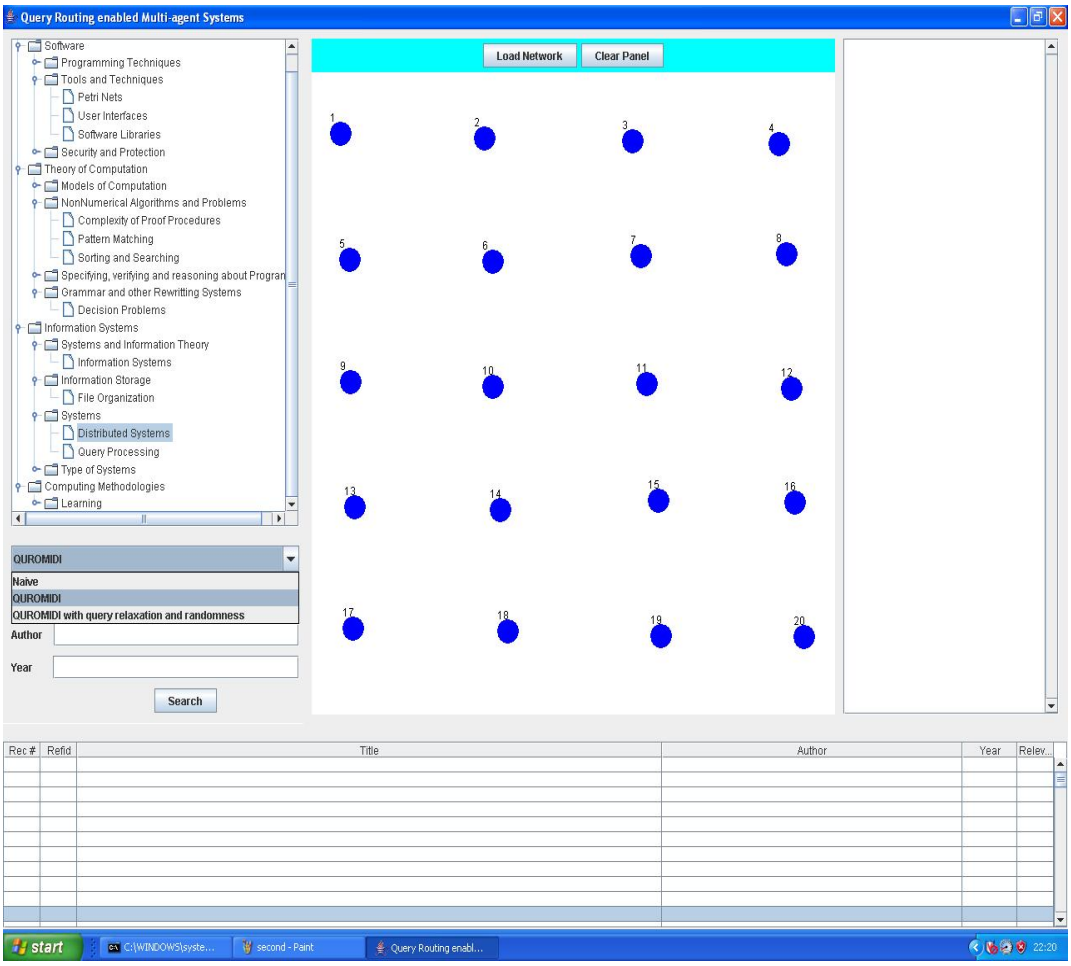



**Figure 12: Query routing algorithm selection**

After algorithm selection the user will enter the keywords of bibliographic reference; he/she wants to search. We assume that the user is aware of title of main keywords of the title; he/she is going to search. After entering these keywords, he/she will press the search button as shown in Figure 13:

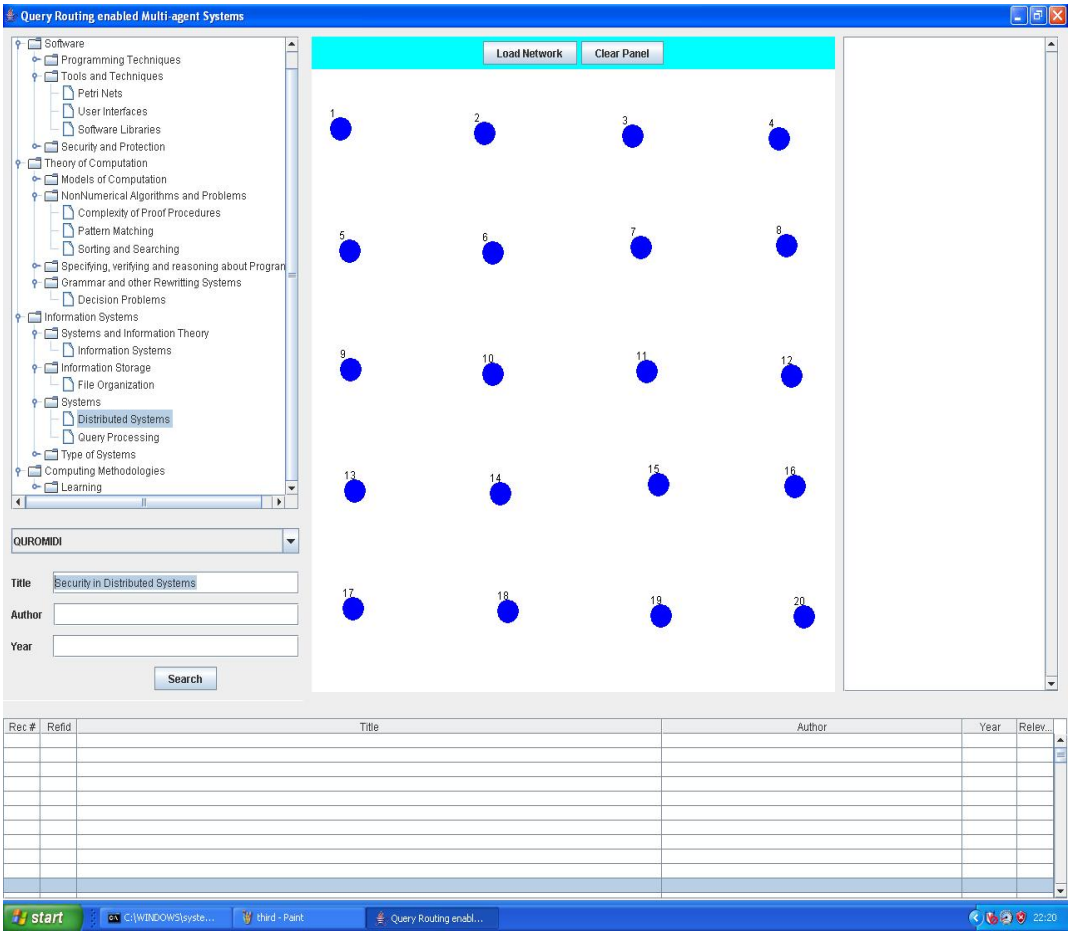



**Figure 13: User enters query keywords**

The first agent will be selected on the basis of the topic selected in the ACM tree. It will perform local knowledge base lookup, forward the query to appropriate known agents, receive answer messages from different agents in the network and display the results in the results table as shown in Figure 14. All the agent communication will be displayed in the message board for the user information.

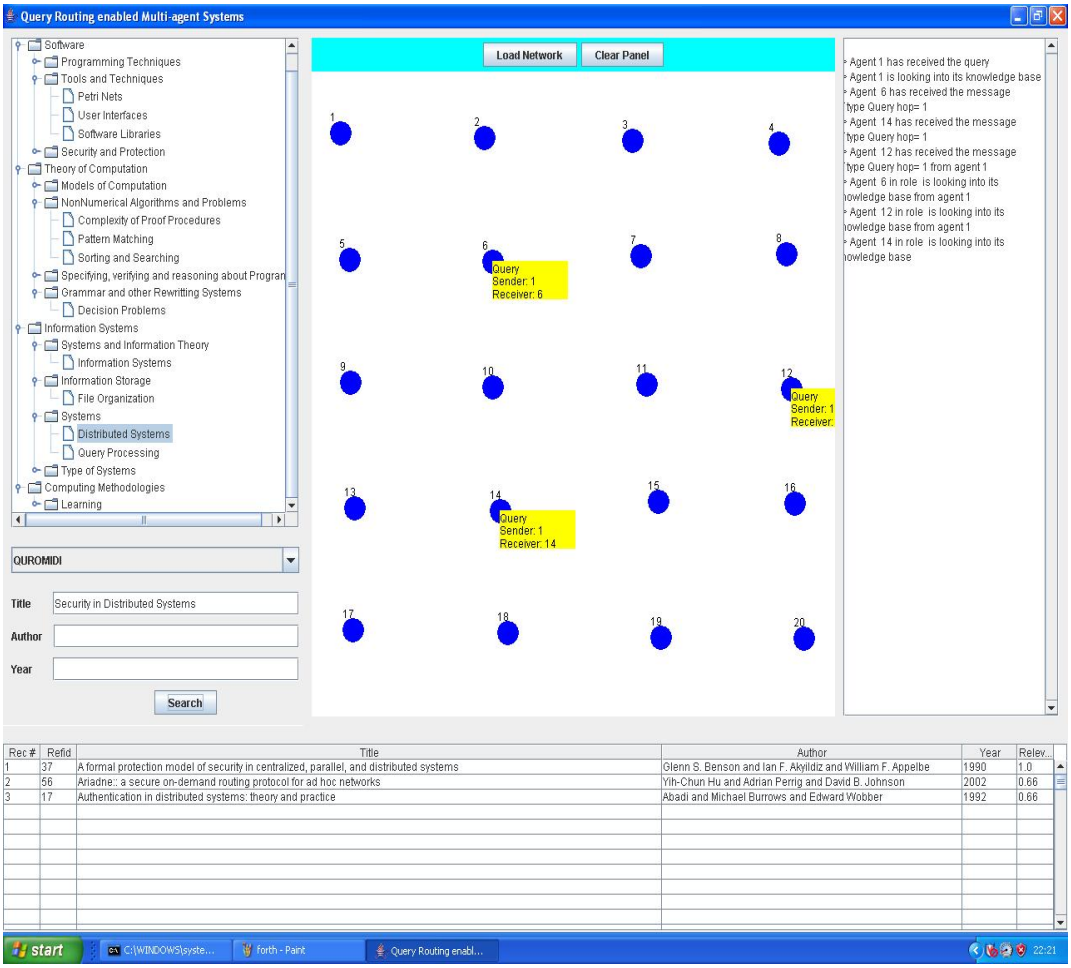



**Figure 14: Results are shown in the table, message board shows agent communication**

The user can see details for any retrieved results by clicking on it. When a user clicks on a particular result a pop up window is opened displaying the complete details for a bibliographic references in Bibtex format as shown in Figure 15:

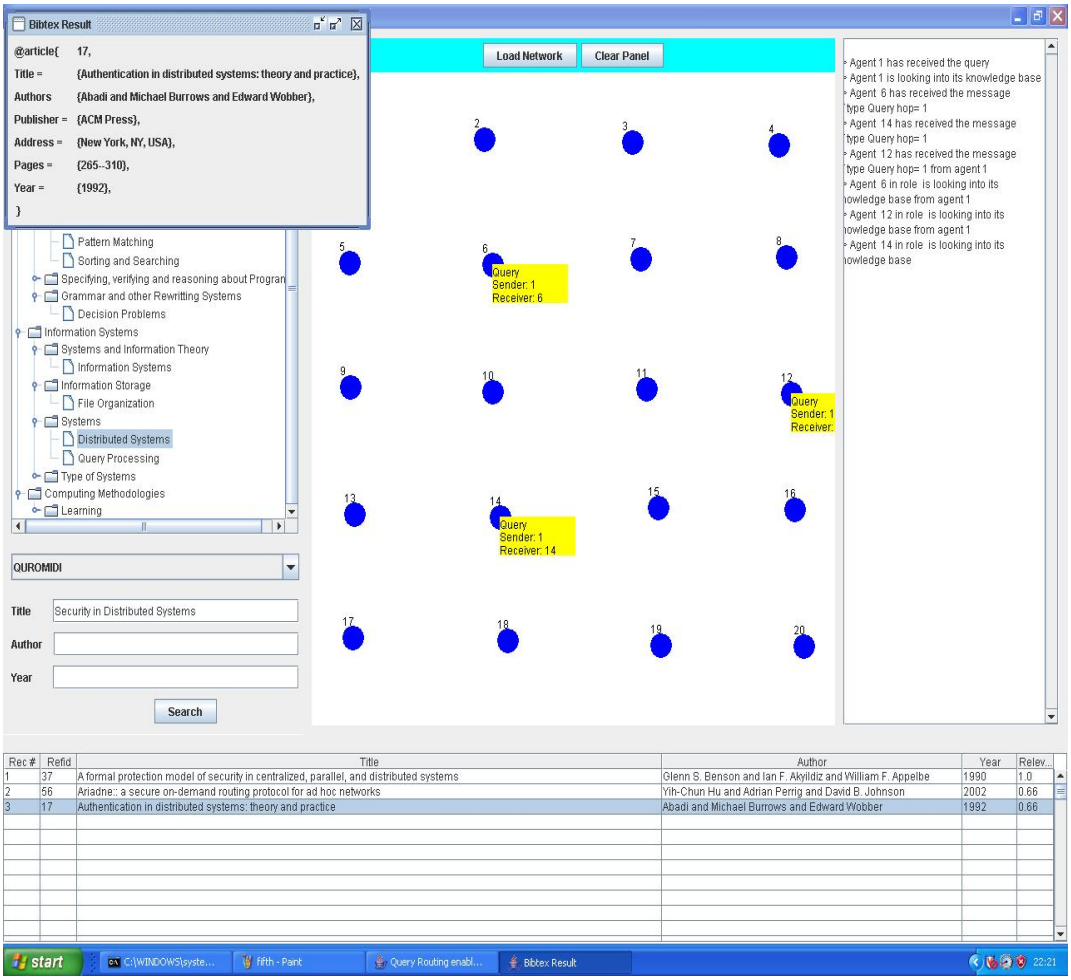



**Figure 15: Result details in Bibtex format**

# Chapter 5

# System Description

This is a simulation based system which simulates intelligent query routing in an agent-based P2P system. It has been based on the social metaphors of daily life interactions among human beings. For learning purposes a *Lazy learning* approach has been used. The chosen scenario is searching for Bibtex references. The ACM topic hierarchy has been used as an ontology among the agents to share the resources semantically. Standard information retrieval parameters recall, precision, and network load have been used for evaluation.

## 5.1. Social Networks

Almost every one of us is a part of some social networks in our daily lives. Our colleagues, friends, and family members are examples of these social networks. We are also connected to other networks, directly or indirectly, through these networks. These smaller networks constitute larger networks which constitute communities (as shown in Figure 16) [30]. In our daily life social networks [7]:

- We usually know about other's knowledge.
- Sometimes, we know about other's knowledge or expertise without having any direct communication just based on our observation.
- Knowledge about others' knowledge or expertise is quite flexible, could be updated any time, adjustable according to the newly found observations.
- Based upon all this information, we can easily find the most appropriate person to answer a particular query related to her/his expertise.

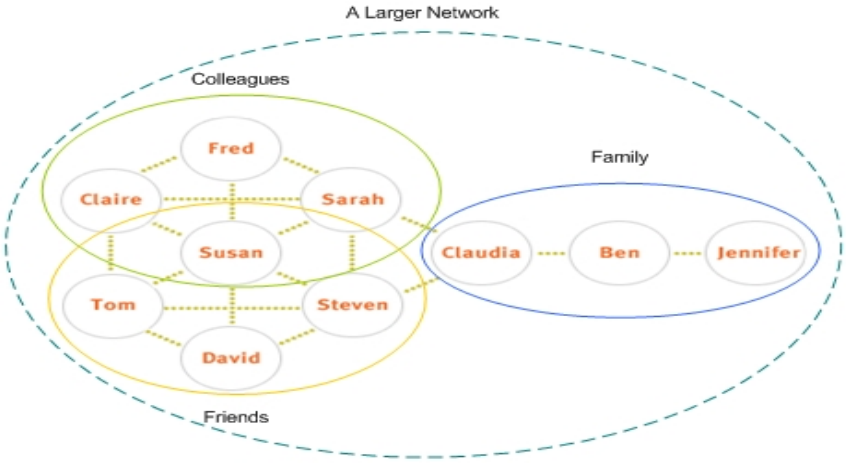



**Figure 16 : A larger social network consisting of some smaller networks**

### 5.1.1. Social Metaphors

P2P systems are decentralized computer networks which mimic social networks to some extent. In social networks the most important task is to identify a person, among the poo[4]l of several possible choices, who could satisfy our queries with appropriate answers. As Peter Morville points out [31]:

"We use people to find content. We use content to find people. Success in the former requires we know what other people know and who other people know. Success in the later demands good search, navigation and content management systems. We might also think of the documents themselves as "human surrogates," representing the knowledge and interests of authors. And of course, we humans also serve as surrogates for one another." As shown in Figure 17 [31].


**Figure 17: Social metaphor**

[4] By "Social Networks" we mean group of people who have professional relationships rather recreational.

Our work is based on the algorithmic form **QUROMIDI (QUery ROuting by Memorising Information about Distributed Information)** of these social metaphors[5] learned from such social networks [7, 8].

- Usually a question is asked to the person whom one assumes is the most appropriate to answer or to refer someone else concerned.
- A person is considered as knowledgeable in a certain domain if he/she has answered previous queries successfully.
- A question is asked to a person, in the absence of a domain expert, whose expertise is similar to the domain of query.
- It's very rare that people are knowledgeable in the domains outside of their domain of expertise.
- There is no absolute scale to measure the performance of others knowledge, rather it is considered to be relative to one's own knowledge.

## 5.2. Some Important Concepts & Terminologies

In this section, we will go through the definitions of some important concepts and terminologies used in this system. This will help the user to understand the overall working of the system more effectively.

### 5.2.1. Agent

Our agent-based P2P network consists of set of agents *A*. Every agent owns a searchable knowledge base consisting of Bibtex information and observations about other agents in the network. In this system we will be using three different terms for an agent depending upon the activity it is performing. *Querying Agent:* one who initially receives the query, *Queried Agent:* one who has been sent query message from the querying agent. It sends an answer message if it finds the answer to the query. *Intermediate Agent:* queried agent or some other agent who forwards the query to its known agents.

5 These metaphors are not result of an exhaustive observation of social networks, there could be exceptions as well

### 5.2.2. Common Ontology

The ACM topic hierarchy has been used as an ontology to give a shared conceptualization of the domain. It has been used to give a semantic description of the '*agent expertise'* and '*query topic'*. It has relations like *subTopic* and *seeAlso.*

### 5.2.3. Expertise

An expertise is a semantic description, of the agent's interest of research, based upon the common ontology *O.* An agent is expert in exactly one field. But there could be more than one agent with the same expertise.

### 5.2.4. Queries

Queries are initiated by a user. They consist of the keywords about a bibliographic reference, mainly from the title or related field, a user wants to search for.

### 5.2.5. Topics

As every agent has a unique expertise, each query has a topic which is a semantic description of the related field of the query, based upon the common ontology.

### 5.2.6. Maximum Hop Count Limit

The number of agents through which a query is passed between its source and destination is known as a hop count. For example, if agent A is the querying agent, it forwards the query to agent B, this makes hop count = 1. Now agent B forwards the query to agent C, which will make the hop count = 2. The maximum hop count limit defines that how many hops a query can be forwarded. The maximum hop count limit in our system is set to 2. Continuing the above example, agent C can forward the same query to agent D but agent D cannot forward this query because now hop count = 3.

### 5.2.7. Query Message

When an agent receives a query from the user, it checks the query against its local knowledge base and forwards the query to its best two (out of four) known agents. These

known agents repeat the same process but they can only forward the query to their known agents if the maximum hop count is not reached yet (which is 2 in our system). This type of message is called a query message. A query message contains the Id of querying agent (which is used by the replier agent to send the answer message directly to the querying agent), Id of intermediate agent (this is added after the first hop), message type (query), query, hop count etc.

### 5.2.8. Answer Message

When an agent receives a query message from another agent, it searches its local knowledge base and if it finds the answer it sends an answer message containing the answer to the querying agent otherwise no message is sent. If the querying agent does not receive any answer from the queried agent assumes that it has no answer to this query and ranks it accordingly by invoking index rank algorithm (see algorithms for details). Answer message contains the Id of querying agent, intermediate agent's Id (if query is not directly sent to it by querying agent), message type (answer), query, hop count etc.

### 5.2.9. Query Routing Mechanism

To route the query to the appropriate agents in the network *hybrid* query routing mechanism (see section 3.5.3 for details) has been implemented. For each routed query a query route is built to avoid cycles. Results are retuned directly to the querying agent (one who initially received the query).

### 5.2.10.Semantic Similarity Value

The similarity function (see section 3.3 for details) is used to find the similarity value between the query topic and agents expertise. Similarity value could also be calculated between the expertises of two agents to judge the similarity of expertise. Its value can float in the range of *zero* to *one. Zero* indicates no similarity at all while *one* indicates exact match. The threshold value for this system is 0.50.

### 5.2.11.Known Agents

In this system we have simulated a *twenty* agent network. Each agent is acquainted with four other random agents. This relationship is known as '*knows*' and these agents are called '*known agents'*. For scalability tests it was upgraded to a network of 40 and 60 agents respectively.

### 5.2.12.Semantic Topology

This work is not aimed to address the issues of network topologies rather it is focused on semantic topologies among agents which are independent of the underlying network topologies. Semantic information about the expertise of other agents is the building block of the semantic topology. Expertise based selection of known agents along with the semantic topology lays down the basis for intelligent query routing.

### 5.2.13.Discovering New Known Agents

Initially agents have randomly assigned known agents. But they can discover new agents through their known agents' known agents. The agents which succeed in answering the query are added into the known agent group. For this purpose several factors are considered. Such as, similarity value of expertise, number of records retuned network awareness level etc. A simple scenario of agent discovery is shown in Figure 18, where hop count represents the number of times a query has been forwarded.

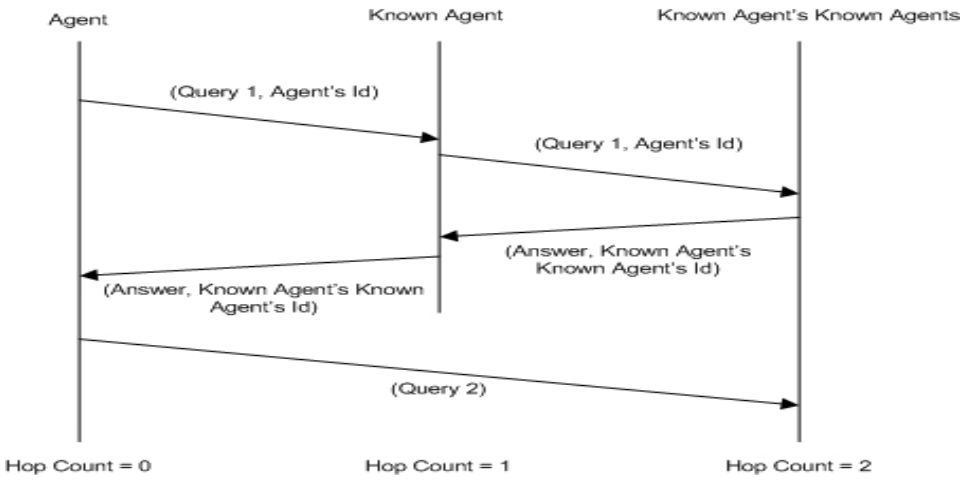



**Figure 18: A simple agent discovery scenario**

### 5.2.14.Naïve Base Line Algorithm

A naïve base line algorithm has been used as base line to compare the results with QUROMIDI. This simply routes the query to its two agents out of its four known agents selected at random.

## 5.3. System Overview

Based upon the above mentioned social metaphors and with the objective of conducting a semantic search in a distributed environment QUROMIDI enables the agents to play the same role as a person in a social network. In our system each agent is expert on one subject. This subject is one of the topics in the ACM topic hierarchy which is being used as an ontology (for details see section 3.2). Every agent maintains its own local knowledge base consisting of facts about network queries constituting '*expertise-based knowledge*'. This knowledge is used to answer the incoming queries locally or to route them to appropriate agents in the network.

The facts about the observed network traffic are stored in the form of a '*semantic reference index*'. These indices are created, maintained, and used in a *lazy learning* style. As QUROMIDI saves the information about other agents with any processing and uses it later for decision making when a new query comes (for details see section 3.1). These

semantic reference indices enable an agent to cooperate with other agents in the following ways:

- On the reception of a query the best agents are selected to route the query to, by the dynamic agent selection algorithm (see section 5.5.2 for details).
- This algorithm uses the information stored in the form of indices to select those agents who have successfully answered similar queries previously. Such agents are called 'content holders' (see section 5.4 for details).

- If no content holders could be identified the best counsellors are selected. Counsellors are those agents who have been involved in similar queries as intermediate agents. Now if a query is routed to them they can route it to the querying agent (the one which issued similar queries last time, not the current querying agent) who can route it to the appropriate content holders (see section 5.4 for details).

- If neither content holders nor counsellors could be selected then net-worker agents are selected. These are the agents which have been actively involved in routing queries on a wide range of similar topics. These agents relax the query against their indices by invoking query relaxation algorithm. In this process of query relaxation some agents are selected which have never been encountered for such queries previously. This involvement of previously unknown agents is called random contribution or randomness. This randomness is sometimes very helpful to avoid over fitting in the selection of agents.

To summarise,

- The algorithms select (at most) two best agents based upon the similarity in query subject and agent expertise along with other performance related facts.
- Route the query.
- Stores and update the facts about the responses of the agents whom queries have been sent.

Our approach is contrary to the approach of advertising the agent expertise upfront. This approach has been implemented by the other member of this research study (as

mentioned earlier in chapter 1). In our approach agent expertise is extracted from the network observations which help to build a dynamic semantic topology by adapting to user queries. Our algorithm claims the following advantages:

- A more effective and efficient query routing mechanism could be achieved by adapting to user queries over time.
- Is adjustable with the changes; and yields a dynamic semantic topology.
- It improves the process of resource discovery by the use of previously learned semantic information.

Figure 19 shows an overview of the system. It presents the architecture of an agent which is a part of this agent-based P2P system. It shows the overall working of the system i.e. from receiving the query to displaying the results to the user. The cloud represents other agents in the system. They also have the same architecture. They search the query into their local knowledge bases, forward it to their known agents (if hop count allows) and return the answers to the querying agent.

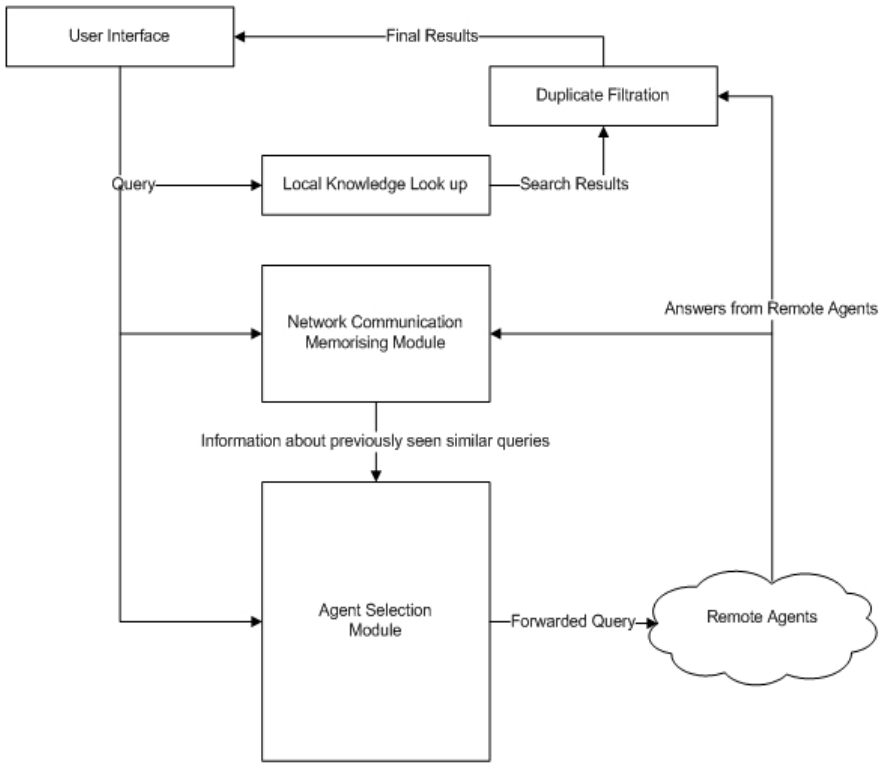



**Figure 19: System overview**

### 5.3.1. Basic Assumptions

We have used the following assumptions in the development of this system;

- Our users are computer science researchers, they are not naïve users.
- They are well aware of the ACM topic hierarchy and know how to use it.
- First agent will be selected on the basis of the topic, a user selects from the ACM topic hierarchy (given in the form of drop down tree in the user interface). So user will not select any topic which is irrelevant to the query entered.
- A query will consist of keywords mainly from the titles of the bibliographic references stored in the knowledge bases of the agents.
- Users will not enter any irrelevant or misspelled queries.
- The number of agents and size of knowledge base of agents is fixed. Agents cannot leave or join the network.
- As we were dealing with the issues of dynamic semantic topologies, it has been assumed that network topology operates perfectly. No network issues have been discussed.
- Communication medium is safe, no threats of unauthorized access. Agents are reliable, no threats of system failure.

### 5.3.2. Chosen scenario

The scenario chosen to test our simulation is to share Bibtex information among researchers. Bibtex is one the most famous formats of bibliographic references being used in the research communities. Researchers keep and maintain lists of bibliographic data. They have to put extra efforts into it and even then they do not have a comprehensive overview of this collection as there could be hundreds of kilobytes of information in dozens of Bibtex files. They do want to share it with other researchers but the hurdle is the time they have to dedicate in doing all the maintenance. This scenario is very well suited for our simulation. As we assume that each agent represents a researcher which has its own collection of Bibtex information in the form of local knowledge base. One can search its own knowledge base as well as over the network in order to get the

required Bibtex information. This scenario has the following features which make it interesting for further investigations:

- Bibtex data has strictly defined basic fields but is flexible to add new ones according to the preferences of the user.
- The most interesting thing is that Bibtex data can never be captured fully under a centralized repository. This is the same issue which we want to address by decentralized repositories. Like DBLP although is a large repository but still does not cover all the related fields (e.g. it covers Databases, AI etc., but not organizational issues of knowledge management)
- Furthermore, it is small enough to be realistic and controllable.

### 5.3.3. A Sample Execution Scenario

At this point, it would be a good idea to have a feel of the execution of the system before moving towards the discussion of technical details. We have already seen user interface details in the previous chapter which would now help the reader to visualise the whole process (It is recommended, for those who have not seen these details, to see 4.2 before reading this scenario).

When a researcher wants to search a particular bibliographic reference, first of all, she will have to select a topic from the ACM topic hierarchy available in the form of a drop down tree. This selected topic should be an exact or relevant topic to the field of the query. After selection of topic the user enters the query consisting of keywords mainly from the title of the articles she is looking for. Then she selects algorithm type from the drop down list (naïve, QUROMIDI, QUROMIDI with query relaxation and randomness). Later results will show that the last algorithm gives the best results, so for this example we assume that user has selected this last option as well and presses the search button.

At this point, query is passed to the first agent (querying agent). Its selection depends upon the topic selected in the ACM tree so it is important that this topic should be similar or exactly the same as that of the query. Now the querying agent will search into its local knowledge base. It also selects the two best agents from its known agents to route the

query to. On receiving query message from the querying agent, they will search their local knowledge base. They will send answer messages only if they have found some relevant results otherwise they will not send any message. By now hop count has become two, which is the maximum limit in our case, so these agents can forward query messages to their known agents. But these known agents cannot forward query messages to their known agent as the hop count would be three. These agents will only look into their local knowledge bases and send answer messages directly to the querying agent if they found any relevant answers. Figure 20 shows an execution scenario describing query search in a sequence diagram.

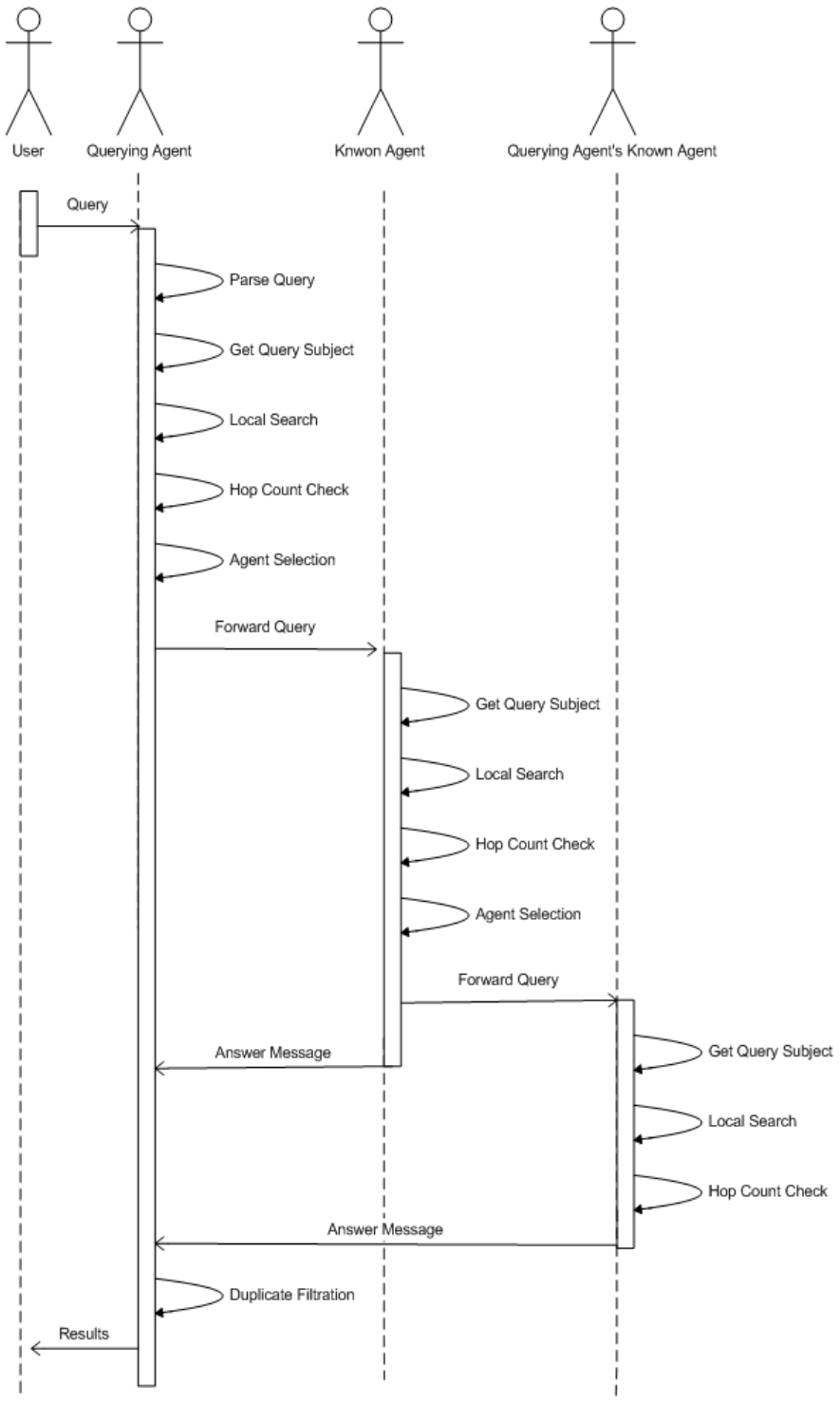



**Figure 20: A sequence diagram describing the query search**

### 5.3.4. Querying Agent Selection

After the initial steps:

- Selection of a relevant topic from the ACM tree
- Selection of search algorithm
- Entering the query keywords

When a user presses the search button the simulation selects the first agent based upon the topic selected by the user in the ACM tree in left panel. For this purpose, a semantic similarity function is used. This function calculates similarity value between the topic selected by the user and the agents having similar expertise. The agent with the highest similarity value is selected as querying agent and the query is passed to it. This agent now will parse the query and search for it in its local knowledge base using the search mechanism described below. It displays the results to the user if it has found any. It selects its two best known agents and route the query to them.

### 5.3.5. Search Mechanism

In this system, we have used very simple search mechanism because the focus of the system is to explore the area of resource discovery by query routing instead of implementing an efficient search mechanism in distributed information repositories. Our search mechanism uses a keyword matching technique. Each agent's knowledge base contains keywords associated with each of the bibliographic references. These keywords are matched with keywords from the query. we have the assumption that our users are computer science researchers, they will use most suitable words as much as possible to avoid irrelevant results (see section 4.3.3 for details).

When an agent receives a query, the query parser parses the query into individual keywords and eliminate any propositions, articles like 'for', 'to', 'a', 'an' etc. Furthermore, we have divided keywords into two different categories *I) normal keywords II) weighted keywords.* Normal keywords are those words which are used quite commonly in the titles and keywords of the bibliographic reference whereas weighted keywords are those words which do not occur very commonly in the titles or keywords of the published papers.

Consider the following titles which have both normal and weighted words:

- Parallel methods in programming
- A new paradigm for functional programming
- Using finite state automata
- A brief history of cellular automata

Here programming is a normal category words as it is appearing in the titles of first two records. Similarly ‘automata’ is also a normal word. Whereas, other words like functional, parallel, new, paradigm, using, finite, state etc. are weighted words. So, we can say that normal category words mostly represent a particular topic or field like programming and automata are subtopics of programming techniques and models of computation respectively. The reason for dividing them into these two categories is that normal category words occur quite frequently in the titles and keywords of the published papers, if we do not distinguish these words they can introduce many irrelevant results.

For example, if a query consists of the keywords 'Typecasting in functional programming'. Using a query mechanism with a 50% relevance threshold (i.e. at least 50% keywords from the query must match the keywords associated with the particular bibliographic reference to include it as a relevant answer to the query, see section 6.1.2 for details) and without these categories could include irrelevant results like 'Techniques of parallel programming', 'A new paradigm for functional programming' etc. But by using our search mechanism with these categories irrelevant results will not be included. Because an agent will not search its knowledge base for a single normal category word instead it will search for a normal word in combination with a weighted category word. This will ensure that no results are being considered as relevant without having weighted keywords in it.

### 5.3.6. Duplication Record Filtration

As our system is based on a decentralised framework, there is no single indexing authority who could index the records to eliminate duplications. In our application every

agent maintains an index of its own knowledge base. The querying agent receives answer messages from a number of agents in the network. There are fair chances of getting duplicate results in response to a single query. It is the responsibility of the querying agent to ensure that no duplicate results are presented to the user.

For this purpose, a querying agent uses a simple mechanism; it compares the title of every new result with each of the existing results. If any two results have the same title then those are considered as duplicate results. In our system duplicate results are not presented to the user, they are simply discarded. Only one result is presented to the user. However, there could be some exceptions to this procedure such as:

- If two results are exactly the same except a conflicting value in one field. For example most of the time it is the name of the publisher, as people sometimes use acronyms instead of full names. So, if the querying agent receives two results with the title 'Typecasting in functional programming'. One of them has ‘ACM Press’ and the other one has ‘Association for Computing Machinery Press’ as publisher. We humans know that these two are the same thing but the agent cannot know so it counts the number of words and prefers the detailed one.

- In another case, there could be two results with same title but one has some missing information. In this case both of them will be merged to get one complete result. For example one result has missing information for its year field while the other has missing information for its pages field. Both of them will be merged to produce one complete result. However, if only one of the results is missing some information while the other is complete the other one is given preference.

## 5.4. Network Architecture

As mentioned earlier, our system is based upon the metaphors learned from social networks (see section 5.1.1 for details). These metaphors divide our network into virtual layers which help to recognise and rank the known agents according to their capabilities to answer a given query. We are dealing the semantic network not the actual physical underlying network issues (see section 5.3.1 for details about assumptions). Subsequent

sections will describe these layers in detail. Figure 21 shows graphical view of these layers [8].

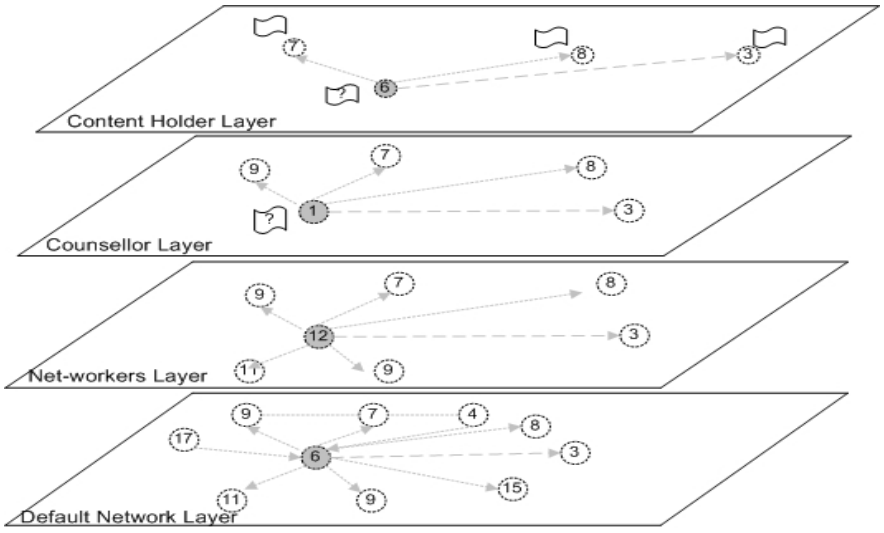



**Figure 21: A layered view of the network**

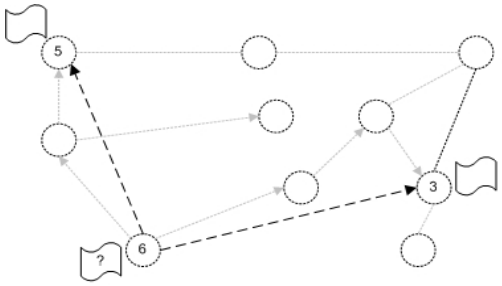
### 5.4.1. Content Holder Layer

Initially an agent has no information about the performance of other agents who have been assigned to it randomly. So, it finds its answers by communicating at a lower layer and flooding the query to the two best known agents. On reception of a successful answer the querying agent creates a '*semantic content holder reference index*' about the replier agent. This replier could either be one of the already known agents or a remote agent (i.e. one of the known agents' known agents). This reference index includes *(queryId, query, queryTopic, agentId, queryHits, role, timeStamp)*.

Where, *queryId* helps to uniquely identify the query, *query* is the routed query, *queryTopic* is the topic assigned by ontology, *agentId* is the Id of the replier agent, *queryHits* is the number of the returned records, role specifies that this agent is *content holder* for this query (the replier agent), *timeStamp* tells about the time, index was created or successfully updated. From now onwards, subsequent queries are checked against this *content holder* index. If a suitable *content holder* cannot be found from this collection of

reference indices and other indices are also not present, then the agent again communicates through lower layers and repeat the whole process to update these indices.

For example, consider the scenario in Figure 22 where agent 6 creates two *content holder* reference indices agent 3 and agent 5 as these agents have the answer for the query sent by agent 6 [8].



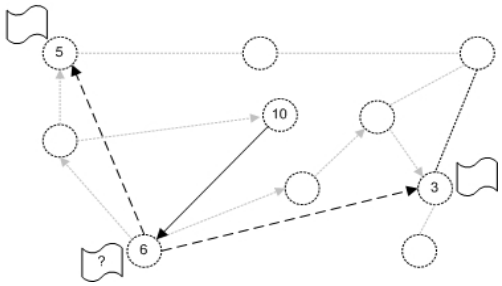




**Figure 22 : Content holder index creation** **Figure 23 : Counsellor index creation**
**where a dotted black line represents content holder index and a solid black line represents counsellor index creation. Transparent lines represent other messages. Tiny leaflets represent agents which have the content and the one with a question mark in a leaflet represents the querying agent.**

### 5.4.2. Counsellor Layer

A *Semantic counsellor reference index* is created at each agent that receives a query from the querying agent. Counsellors are those agents who have been involved in similar queries as intermediate agents. Whereas, counsellor reference index is a semantic reference to the querying agents who have been issuing queries previously. These indices are used by the intermediate agents to route the queries to the agents who have issued similar queries previously. Intermediate agents have the assumption that these querying agents would have received answer for their queries. Now if a query is routed to them they can route it to the querying agent (the one who issued similar queries last time, not the current querying agent) who can route it to the appropriate *content Holders*.

This index includes (*queryId, query, queryTopic, agentId, queryHits, role, timeStamp)*. Where, *queryId* helps to uniquely identify the query, *query* is the routed query, *queryTopic* is the topic assigned by ontology, *agentId* is the Id of the querying agent who issued this query, *queryHits* is the number of the returned records but here we put '-1' as the counsellors cannot get to know that how many records have been sent as replies are

sent directly to the querying agent, role specifies that this agent is counsellor for this query (this role is not related with the agent Id, rather it specifies who can use this index. So it means that an intermediate agent can use a counsellor index because it was the one who received the query from this (*agentId*) querying agent), *timeStamp* tells about the time, index was created or successfully updated.

For example, consider the scenario in Figure 23 where agent 10 creates a counsellor reference index for agent 6 as the query was routed by agent 6 [8].

### 5.4.3. Maintenance of Content Holder and Counsellor Layers

An agent cannot store reference indices about every remote agent. We have to set a limit which is 6 in our system. That means an agent can have information about two new agents in addition to already known agents. Once this limit is reached, agent has to decide which indices to keep and which to delete. This process of index maintenance, according to the criteria mentioned below, ensures the development of a dynamic topology. Because, by following this procedure, agents will keep indices only for those agents which return the most suitable answers. So, after a certain time of learning the network communication, all the agents would be of similar expertise. This will create semantic clusters of the agents sharing similar expertise in the network. For example consider Figure 24 and 25 which shows initial random topology and semantic topology evolved over time respectively. These figures have five different coloured dots. Each dot represents an agent. Colour represents an expertise. Agents with the same colour are having same or similar expertise. Initially as in Figure 24 they are arranged in random order with respect to colour (a random topology). But they evolve a semantic topology over time and form semantic clusters sharing similar expertise as shown in Figure 25.

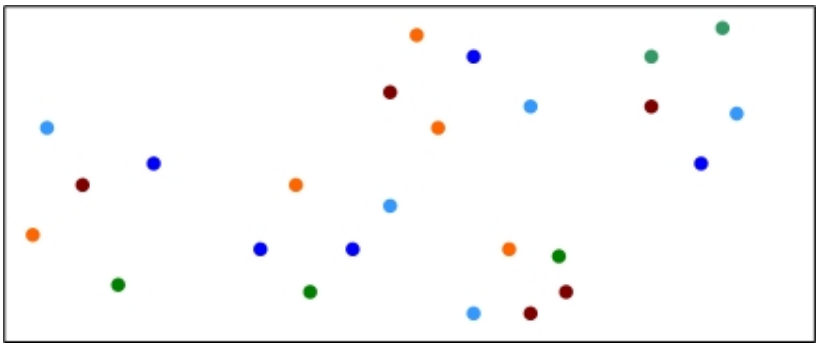


**Figure 24: An initial random topology**

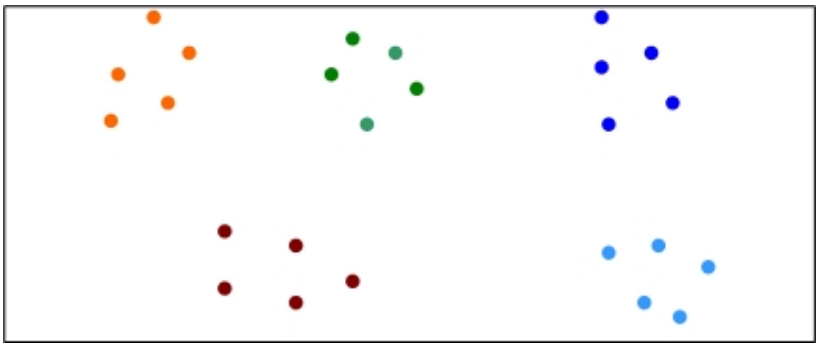


**Figure 25: A semantic topology evolved over time**

The following are the criterion used to maintain the indices:

***Similarity Value:*** Indices are ranked according to the similarity value between the remote agents' expertise and the expertise of the agent holding the indices. The higher the rank is the more are the chances to keep an index.

***Query Hits:*** Query hits are also used to rank the indices. An index with a higher query hit value is preferred over the index with a lower query hit value.

***Reach ability:*** Indices are also ranked according to its index type (i.e. content holder, counsellor). Content holders can be reached within one hop whereas through counsellor at least two hops are needed to get the results. Content holders with an exact match are given a value of 1, with relevant results a value of 1.5 and counsellors are given a value of 2. They are ranked in ascending order.

***Update Interval:*** Indices are also ranked according to the frequencies of their use. Time difference between their last use and recent update is used to rank the indices.

### 5.4.4. Net-workers Layer

A *Net-worker reference index* is a reference to those agents which have been actively involved in routing queries on a wide range of similar topics. These agents are known as *net-workers*, this capability is known as a level of network-awareness. It is judged on the basis of the number of incoming-messages and the number of outgoing-messages. Incoming-messages help to identify the number of distinct sources an agent receives queries from. By outgoing-messages we measure, how successfully an agent shares its resources with other agents. By routing query to net-worker agents, there is a higher probability to find a semantically similar index than to go for the default network.

Each query that is issued in the network contains information about the capability of *network-awareness* of the querying agent. Every agent updates its network index based upon this information. This index is created as *(agentId, network-awareness level). AgentId* is the Id of the net-worker agent while *network-awareness level* indicates the diversity of the agent's links in the network.

Network-awareness level is calculated using the following equation:

$$\textit{Network-awareness level} = \{(1+\textit{Incomming}) + (1+\textit{Outgoing})\}$$

Where incoming is the number of distinct agents who send queries. To get the outgoing in decentralized setting we compute the number of distinct agents with whom agent shares its resources. One is added to avoid zero values.

### 5.4.5. Default Network (Lower) Layer

When an agent is new in the network and has no semantic indices stored in its knowledge base, it uses the default network indices of agents assigned to it randomly.

## 5.5. Algorithms

As mentioned before, this system has been implemented on the algorithmic form (QUROMIDI) of social metaphors learned from the social networks. QUROMIDI is a collection of algorithms which are used together to achieve the overall goal of efficient resource discovery. These algorithms are Dynamic *agent selection, Query Relaxation, Index rank* etc.

### 5.5.1. Protocol Scenario

These algorithms include several steps of execution performed locally and over the network when routing the queries or responding the queries and also while receiving responses. A user initiates a query which is evaluated:

***Locally:*** On the reception of a query first of all it is evaluated against local knowledge base. In this process the search mechanism described in section 5.3.5 is used.

***Over the Network (Routing the Query):*** After evaluation of the query against local knowledge base, *Dynamic agent selection* algorithm is invoked. Its task is to find the best N(where N=2) agents out of the content and counsellor indices to which a query should be routed. If it cannot find any suitable agents, net-workers are selected to relax the query by invoking *query relaxation* algorithm. Query relaxation algorithm uses the mechanism described in section 5.5.2. Original query is routed to the selected N agents.

***Over the Network (Responding the Query):*** As an agent receives a query it tries to answer it locally and stores an index of *counsellor* type for the querying agent. An answer is retuned only if there is one found, otherwise no response is sent. These responses are sent directly to the querying agent. But the *Index rank* algorithm is always invoked by the querying agent for every agent to which the query was sent even if they have retuned no answers. The process of routing the query continues until the maximum hop (which is 2) limit is not reached.

***Receiving Responses:*** On reception of the answers at the querying agent an index is created of the type *content holder* for every replier agent. The answers are evaluated

according to relevance and duplicate records are filtered. Index rank is invoked for every response to update the indices of that particular agent.

### 5.5.2. Description of Algorithms

In this section we will briefly describe of the algorithms used in this system.

***Dynamic Agent Selection:*** The main objective behind keeping index records is to enable the agents to dynamically adapt the topology of the network which results in the form of semantic groups, sharing common expertise. This algorithm returns a set of two best agents to which to route the query. Initially it tries to select the agents from *content* and *counsellor* indices whose expertises have highest similarity value with the topic of the query. Out of these agents, those agents are selected whose similarity value is above a certain threshold (which is 0.50). In case two agents have a conflict because of same similarity value the preference is given to the agent with highest query hits. If no agents could be found or less than the required number then net-worker(s) is/are selected by repeating the same process. They invoke query relaxation to broaden the query. The algorithm has two main tasks I) to ensure the required number of agents have been selected II) to choose some random agents to avoid over fitting. The algorithm terminates if the required number of agents have been selected or the maximum hop count has been reached.

***Query Relaxation:*** This algorithm exploits the assumption that if someone is knowledgeable in one field, she might have knowledge about other closely related fields as well. So, as we know that agent's expertise is also a topic from the ACM topic hierarchy it calculates the semantic similarity between the immediate parent topic or parent topic's parent topic of the query with agent's expertise successively to find best N (where N=2) agents. A query is relaxed until suitable agents are found or the ACM root is reached as it is not possible to further relax the query topic. The selected agents are then passed the original query. Results discussed in evaluation chapter will show that QUROMIDI shows best performance with the query relaxation option.

***Index Rank:*** This algorithm is invoked by the querying agent to rank the indices on reception of an answer. It is also invoked even if an agent has not sent any answer which has been sent query by the querying agent. This time it is invoked with the assumption that agent has not the answer to this query. Index rank algorithm uses the criteria mentioned in section 5.4.3 to rank the indices.

# Chapter 6

# Evaluation

This chapter describes the evaluation and discusses of the results obtained during the evaluation process. Evaluation is always considered to be the most critical part of any results oriented task. The following are the two most important objectives of an evaluation procedure:

- To assess how efficiently and effectively a program achieves its goals.
- To improve the current system by discovering new possible functionalities or design of the system.

## 6.1. Evaluation Criteria

This section defines the input/output parameters and the criteria used for the evaluation of our system.

### 6.1.1. Input Parameters

The following are the input parameters which are critical to the performance of any P2P based system [32]:

***Number of Agents:*** This number represents the size of the network i.e. how many agents are there in an agent-based P2P network. This number is used to access the scalability of the underlying system. In our system, initially we test the system with 20 agents and then for scalability test it was upgraded to a network of 40 and 60 agents respectively.

***Number of Documents:*** This number represents the number of shared resources among the agents which are available on the network. This number can also be used

for scalability tests. In our system, we have 400 documents in a 20 agent network, 800 in 40 and 1200 documents in a 60 agent network.

***Document Distribution:*** Document distribution means, how the documents are distributed among the agents. It could be a random distribution or a selective distribution. With a random distribution an agent could have documents from completely different domains as well as from similar domains. In a selective distribution an agent has the documents mostly from one domain with a small proportion of documents from closely related domains. It is not realistic to state that one agent has documents from precisely one domain because it does not make sense (for example) to state that a document related to Information Retrieval is not relevant to P2P systems.

In our system, we have used selective distribution so an agent's knowledge base mostly consists of the documents from one domain with a small proportion of documents from closely related domains. We have explicitly added documents in the knowledge base of an agent's related field of expertise. This way we make sure that our approach works well to discover resources in a distributed environment.

***Algorithm:*** Algorithms usually have a very strong influence on the performance of the network. We have evaluated our system using three different query routing algorithms.

***Number of Queried Agents:*** This number represents the number of the agents to be selected by the *Agent Selection* algorithm. This number has a strong influence on the network performance as well, because queries forwarded to more agents cause more network messages to be generated.

***Number of Hops:*** This number determines how many times a query could be forwarded. A large number causes the network to be flooded. In our case, we have set it to 2. For scalability tests, we tested it on a value of 0, 1, 2, 3, 4, and 5 subsequently.

| ***Parameter*** | ***Value*** |
|---|---|
| Number of agents | 20 |
| Number of documents | 400 |
| Document distribution | Expertise based |
| Agent selection algorithm | *Dynamic Agent Selection* |
| Number of queried agents | 2 |
| Number hops | 2 |

**Table 2: Standard parameters used in evaluation**

### 6.1.2. Output Parameters

The following are the output parameters which are critical to evaluate the performance of our system:

***Relevance:*** Relevance is a measure assigned by the user (who asks the query) to the retrieved results with respect to the degree of satisfaction of the query. Its value varies in the range of 0 to 1. Usually, it is determined by the number of keywords in the query which match the keywords of the bibliographic reference. For example if there are five words in the query and the two results have been returned. One which has three matching keywords would be of 66% relevance and the other which has all the keywords would of 100% relevance.

***Recall:*** This is the most important parameter which has been the focus of this evaluation procedure. Recall is a standard information retrieval measure. It determines how many documents have been returned out of all the relevant documents in the network, it is defined as:

$$Recall = (Relevant \cap Retrieved) / Relevant$$

(Where Relevant is the set of relevant documents and Retrieved is the set of retrieved documents)

We use recall to access the effectiveness of our system that to what extent it could be used to retrieve the documents in a decentralized environment by memorizing the network communication. Here, a question could be raised that how someone could

know the total number of documents in a decentralized network, the answer is that as it's a simulation where we have the control of the system in order to evaluate the system with respect to a number of variables, so one knows the number of relevant documents in the network.

***Precision:*** Precision is also a standard information retrieval measure. It is used to determine the proportion of relevant documents out of the total retrieved documents. It is defined as:

$$Precision = (Relevant \cap Retrieved) / Retrieved$$

(Where Relevant is the set of relevant documents and Retrieved is the set of retrieved documents)

In our system, we are dealing with exact queries so all the results retrieved are always relevant. So, the precision will always be *one* in our case.

***Network Load:*** It is determined in terms of messages per query. It indicates to what extent network is being flooded by each query. Number of maximum hop helps to determine the efficiency and goal orientation of the query routing algorithm that how fast an answer could be retrieved.

## 6.2. Data Set

Each agent contains a knowledge base of twenty Bibtex records. So, we have sets of 400, 800, and 1200 Bibtex records in 20, 40, and 60 agent networks respectively. These records have been downloaded from the ACM portal [33] under different topics which have been covered in the ACM topic hierarchy used in our system. As already, discussed we have used four distinct first level categories each of which has two to four subtopics and then those subtopics have four to six subtopics. We have selected records under these third level topics which constitute a topic hierarchy for each of the downloaded Bibtex record.

## 6.3. Query Generation

This system has been evaluated against 30 queries for each of the three networks of 20, 40, and 60 agents respectively. The queries consist of keywords mainly from the title of the documents. The evaluation process has been conducted in three phases:

1. In the first phase, we continuously put queries to the system and took the readings for over-time learning.
2. In the second phase, the system was presented with the same queries already answered.
3. Finally, in the third phase, queries were divided into two equal sets. In this scenario, first set was used to train the network (these queries were divided so that they covered almost every topic, so both sets had queries from all the topics). The second set of queries was used to observe the role of *net-workers* using *query relaxation* and the effect of *randomness* in agent selection.

## 6.4. Results and Discussion

The results show that QUROMIDI outperforms the naïve base line algorithm. It acquires a higher recall rate and by using semantically rich agent selection mechanism. Query relaxation by net-workers and the use of randomness in agent selection has also improved the results. Before going to the results, the following are the investigated hypotheses:

- The QUROMIDI algorithm outperforms the naïve algorithm in terms of recall.
- Agents adapt the dynamic topology quickly and adjust according to that.
- Use of net-workers to relax the query can enhance performance.
- Some randomness in agent selection helps to avoid the problem of over fitting.
- Increased hop count effects recall to a certain limit but after that it looses its effectiveness.

### 6.4.1. Sensitivity Analysis

First of all, we had the problem of deciding the standards for relevance threshold (the threshold to declare a result relevant) and the number of queried agents (the number of agents an agent can route query to). The details and the results are as follows:

***Relevance:*** Relevance is decided by the user (see section 6.1.2 for details). There could be many complex techniques to set the relevance threshold but as our system is not for improving search mechanism rather it is to test the effectiveness of resource discovery in decentralized distributed environments, we used simple keyword matching technique to set the relevance threshold. For example, If the user enters a query which consists of four words excluding 'for', 'to', 'of', 'a', 'an' etc. (as these would be excluded automatically by the search mechanism) a result could be declared relevant if it contains two of these words. That means the threshold is set at 50%. To be more precise it could be set to 75%. In that case, three out of those four words have to be in the title or keyword list of the retrieved record. Similarly it could be set to 100% requiring matching all the four keywords entered by the user as a query.

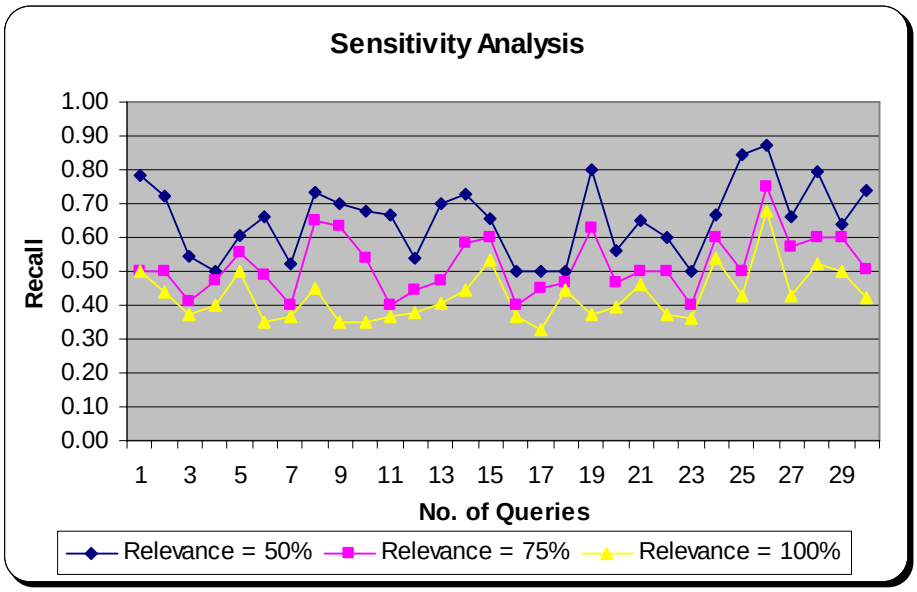



**Figure 26: Sensitivity analysis for relevance threshold**

Figure 26 shows the results of the experiments we performed to decide on a relevance threshold. The x-axis shows the number of queries and y-axis shows the recall for each of the query. We run the experiment three times on the same network for thirty queries with 50%, 75%, and 100% relevance threshold respectively. Recall level decreases as the threshold level increases. It decreased from a good average of *62% for 50% threshold* to *54% for 75% threshold* and from 54% to *46% for 100% threshold*. This much decrease in recall is not affordable for effective resource discovery, unless it's the strict requirement of the user to retrieve just those records having maximum match. However, for our evaluation purposes, we have set relevance threshold to 50% in rest of the experiments.

***No. of Queried Agents:*** No. of queried agents (the number of agents to which an agent should route the query) is a critical issue. It is basically a trade off between recall and number of messages on the network in terms of network load. The number of messages increases with the number of agents contacted for a query in the network. To set a standard for this number we conducted two separate sets of experiments each with thirty queries run on each set. In each set of experiments, the underlying network was using QUROMIDI. It was scaled up to 20, 40, and 60 agent networks respectively. The only difference in these two sets of experiments was the number of queried agents $P_Q$. In first set it was set to $P_Q = 4$ and in the second set $P_Q = 2$.

Finally, we had six different simulation settings having the information as follows:

**Set 1**

- Number of messages per query and recall for intelligent network of 20 agents with $P_Q = 4$.
- Number of messages per query and recall for intelligent network of 40 agents with $P_Q = 4$.
- Number of messages per query and recall for intelligent network of 60 agents with $P_Q = 4$.

Set 2

- Number of messages per query and recall for intelligent network of 20 agents with $P_Q = 2$.
- Number of messages per query and recall for intelligent network of 40 agents with $P_Q = 2$.
- Number of messages per query and recall for intelligent network of 60 agents with $P_Q = 2$.

Upon having a critical analysis of the resulting data presented in Table3 and Table 4, we discovered that recall is almost the same for $P_Q = 4$ and $P_Q = 2$ (in networks with same number of agents) but the average number of messages per query has been decreased by half or more than half of $P_Q = 4$ in $P_Q = 2$. Taking it empirically, recall drops by just 2% in each of the same networks while the average number of messages per query drops by 56%. It shows that $P_Q = 4$ has no advantage over $P_Q = 2$, so we decided to set $P_Q = 2$ for rest of the experiments. The reason why there is no significant difference in *recall,* despite of the fact that one set of experiments is querying double the number of agents, is that it is not guaranteed that all the agents queried would have similar expertise and would be able to answer the queries. That is why, $P_Q = 4$ could not outperform $P_Q = 2$ instead of just flooding the network.

Discussing this chart more fully is important as it delivers some other important facts about the recall. The reader may notice that recall decreases slightly as the networks are scaled up. The reason behind is that it is quite natural to drop the recall as more agents get involved. Because there are more chances to query irrelevant agents resulting in increased number of messages and low recall. However, this difference is not very significant and could be tolerated; furthermore it has been improved as well, by using the *net-workers (with query relaxation and randomness)* mechanism, which we will see later in this chapter. On the next two pages, we can see the charts in Figure 27 and Figure 28 showing the results we discussed so far.

| $P_{Q=4}$ | 20 Agents | 40 Agents | 60 Agents |
|---|---|---|---|
| **Messages per query** | 19 | 18 | 18 |
| **Recall** | 67% | 63% | 61% |

**Table 3: Summary of results for Figure 27, where $P_{Q=4}$**

| $P_{Q=2}$ | 20 Agents | 40 Agents | 60 Agents |
|---|---|---|---|
| **Messages per query** | 7 | 8 | 8 |
| **Recall** | 65% | 61% | 59% |

**Table 4: Summary of results for Figure 28, where $P_{Q=2}$**

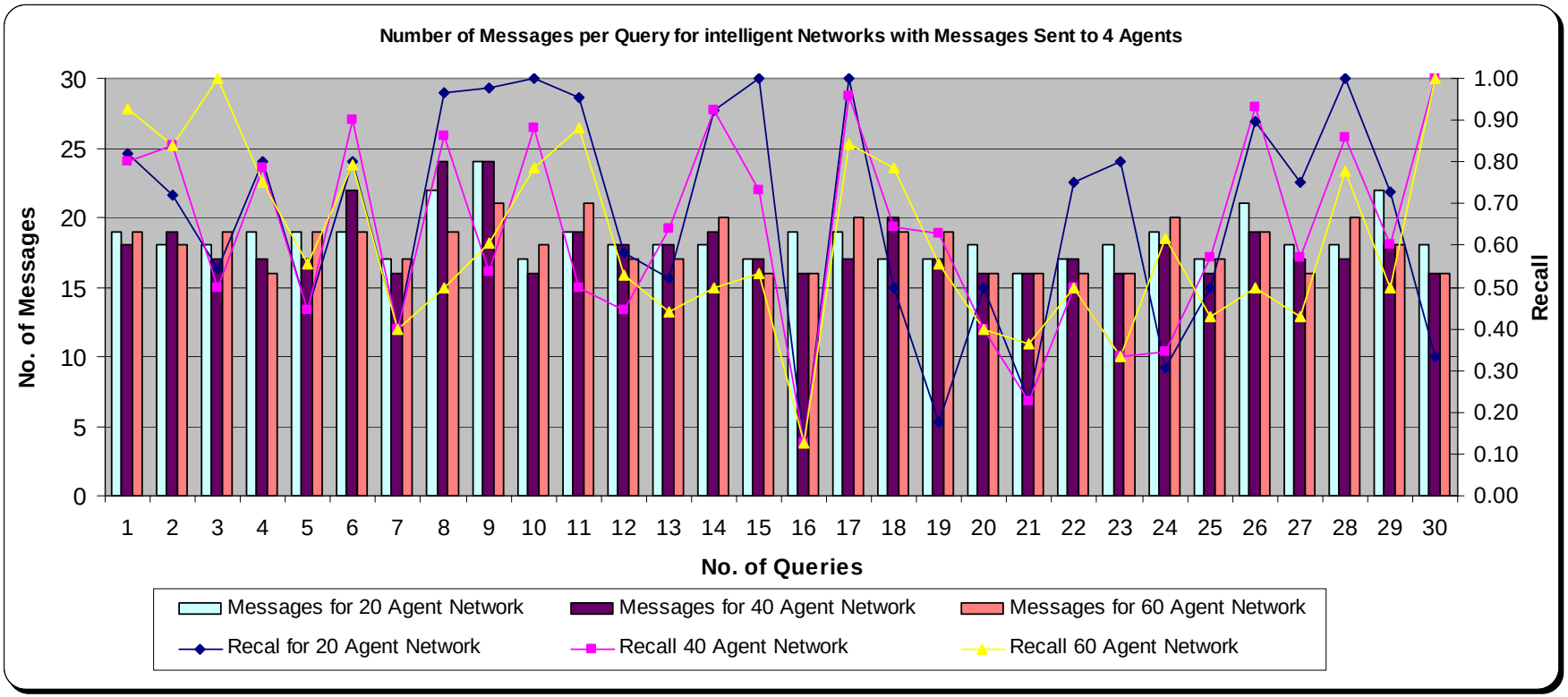



**Figure 27: This figure shows the results of Recall and Number of messages per query for 3 intelligent networks with 20, 40, and 60 agents respectively. While $P_{Q=4}$**

The x-axis shows the number of queries, left y-axis shows the number of messages per query, and right y-axis shows recall for each query.

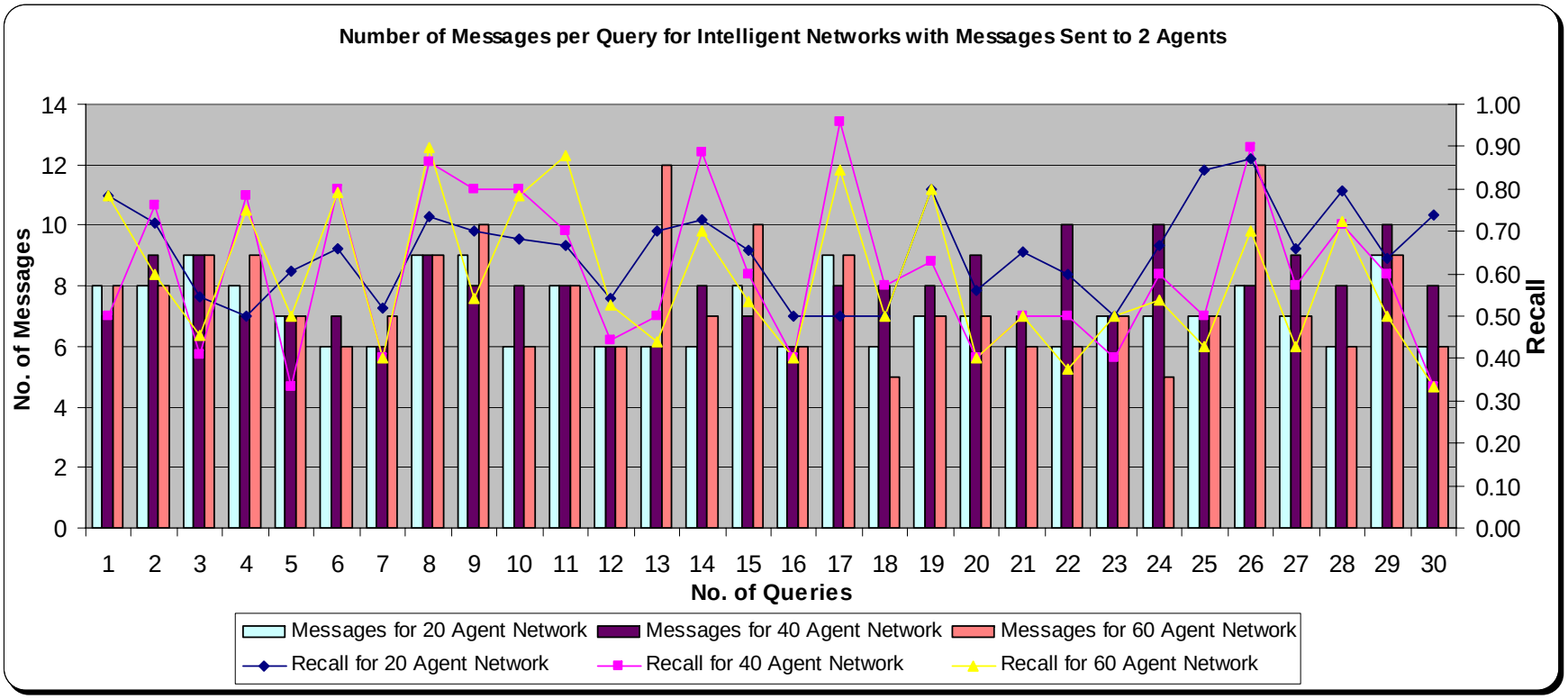



**Figure 28: This figure shows the results of Recall and Number of messages per query for 3 intelligent networks with 20, 40, and 60 agents respectively. While $P_{Q=2}$**

The x-axis shows the number of queries, left y-axis shows the number of messages per query, and right y-axis shows recall for each query.

### 6.4.2. QUROMIDI vs. Naïve Algorithm

In this section we compare the results of the naïve base line algorithm with the results of QUROMIDI to demonstrate that QUROMIDI outperforms the naïve algorithm. Figure 29 and 30 displays the results in each case.

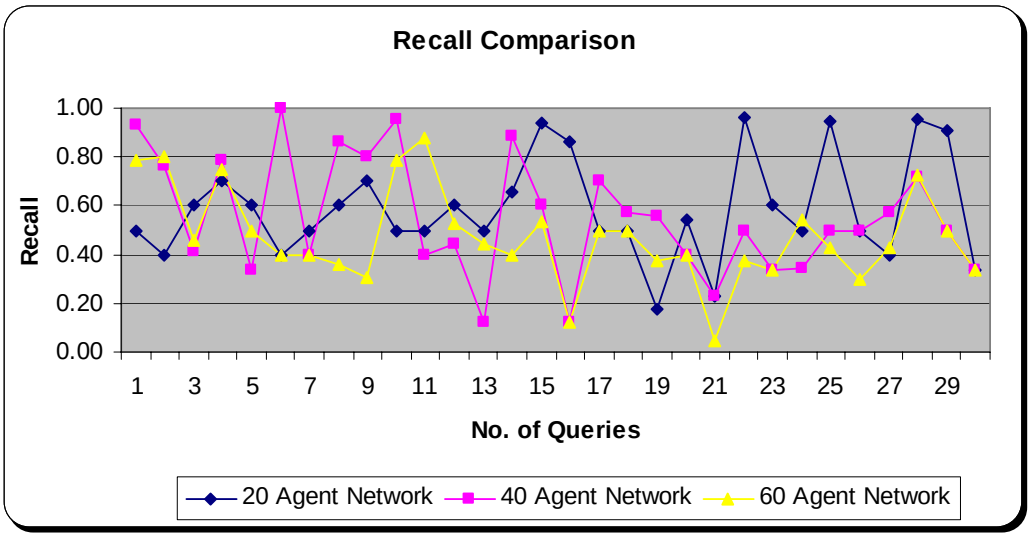



**Figure 29: This figure describes recall comparison for the naïve networks of 20, 40, and 60 agents respectively. With $P_{Q=2}$.**

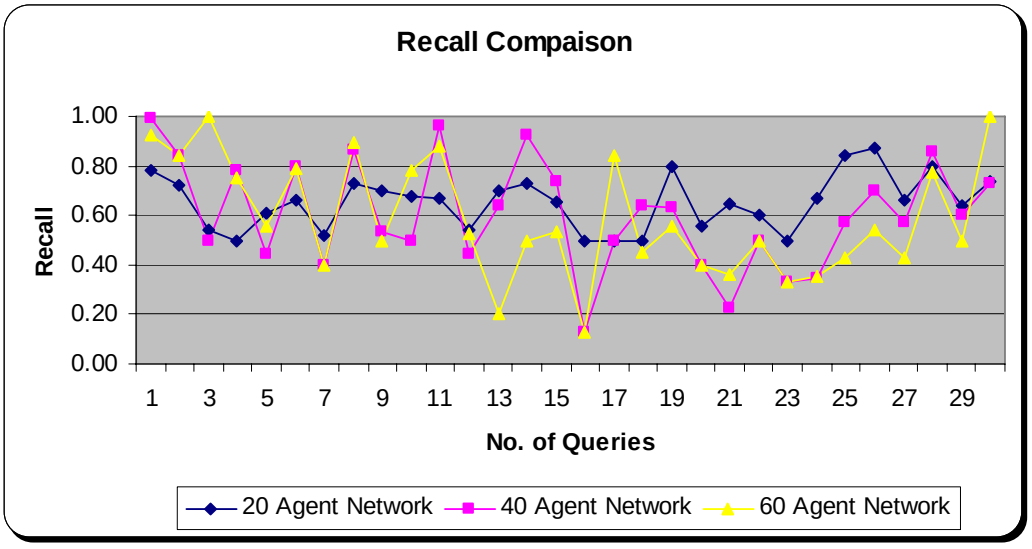



**Figure 30: This figure describes recall comparison for intelligent networks of 20, 40, and 60 agents respectively. With $P_{Q=2}$.**

***Summary:*** The charts above show that QUROMIDI outperforms the naïve algorithm with the following empirical results:

| ***Naïve Algorithm*** | | |
|---|---|---|
| **20 Agents Network** | **40 Agents Network** | **60 Agents Network** |
| 59% | 55% | 47% |

**Table 5: Recall ratio of naïve networks of 20, 40, and 60 agents**

| ***QUROMIDI*** | | |
|---|---|---|
| **20 Agents Network** | **40 Agents Network** | **60 Agents Network** |
| 62% | 60% | 59% |

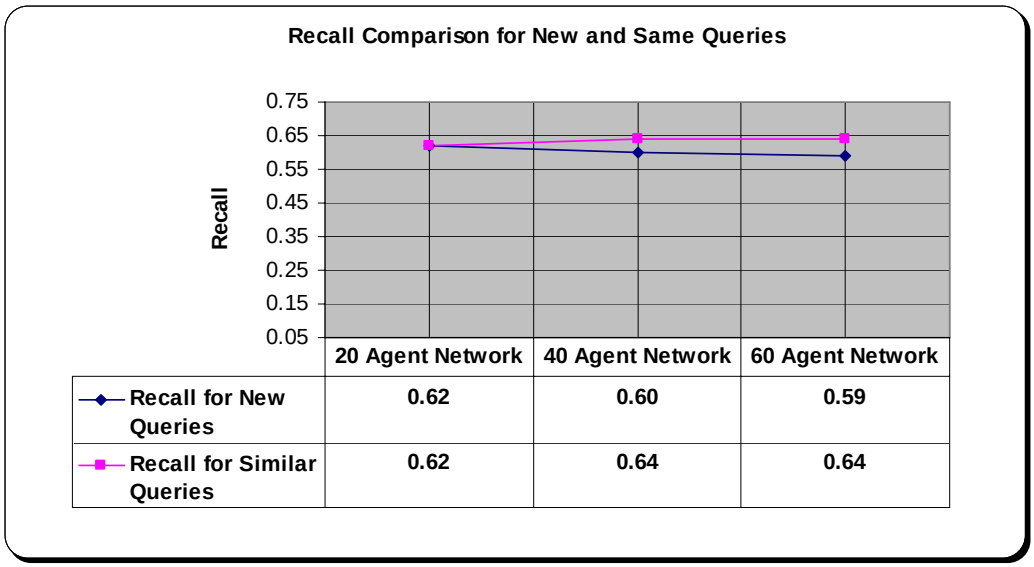
**Table 6: Recall ratio of intelligent networks of 20, 40, and 60 agents**

- The recall in naïve networks drops by 12% from a 20 agent network to a 60 agent network. While in intelligent network the drop is only 3%.
- From another view, we see that recall drops significantly in a 40 agent naïve network as compared to a 40 agent intelligent network and the same is the case for a 60 agent network.
- But a 20 agent network did not show a significant drop in recall. We will discuss the reasons in the critical analysis section (see below).

***Critical Analysis:*** Examining the charts in Figure 29 and 30 and the data in Table 6, one can notice that there is no significant difference in the recall of naïve and intelligent networks with 20 agents. The reason is that a 20 agent network was very small. Since each agent knows 4 random agents, so the probability of querying the right agent was very high. That is why there is no significant difference in the recall ratio. To support this view, the recall ratio comparisons of the other two networks are evident.

### 6.4.3. Recall Ratio of Intelligent Networks for New and Similar Queries

In this section we compare the results and evaluate the performance of the network in answering new queries vs. answering similar queries which have been asked previously.




| | 20 Agent Network | 40 Agent Network | 60 Agent Network |
|---|---|---|---|
| Recall for New Queries | 0.62 | 0.60 | 0.59 |
| Recall for Similar Queries | 0.62 | 0.64 | 0.64 |

**Figure 31: This figure shows recall ratios for intelligent networks of 20, 40, and 60 agents respectively for new and previously asked similar queries**

A summary of the results is given in Figure 31. It shows the recall comparison for intelligent networks of 20, 40, and 60 agents each of which has been tested for 30 queries. The results show that recall rate for new queries drops slightly as the network size increases but on the other hand for similar queries it increases slightly as the network size increases. It shows that agents have learned and have adapted the dynamic topology which enables them to enhance their performance. However, for new queries it is reasonable for the recall rate to drop a little bit as in that case agents are still in the learning process.

### 6.4.4. Comparison of Query Routing Algorithms

In this section we discuss and evaluate the results obtained by applying naïve, simple QUROMIDI, and QUROMIDI (with *query relaxation and randomness*) query routing algorithms. These results show that QUROMIDI (with *query relaxation and*

*randomness*) outperforms both naïve and simple QURMOMIDI. These algorithms have been applied to scaled up networks of 40, and 60 agents as well along with the basic network of 20 agents.

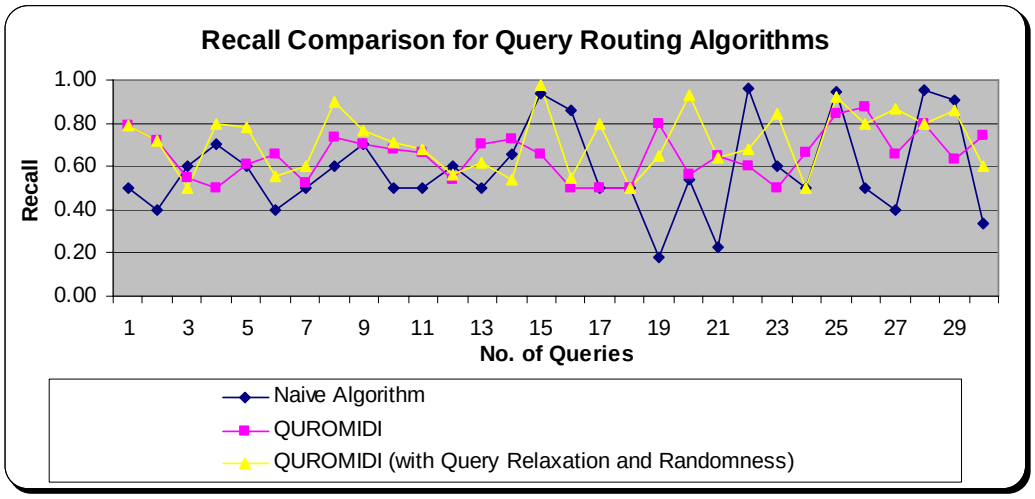



**Figure 32: This figure shows recall comparison of the three different query routing algorithms on a network of 20 agents with 30 queries tested on each**

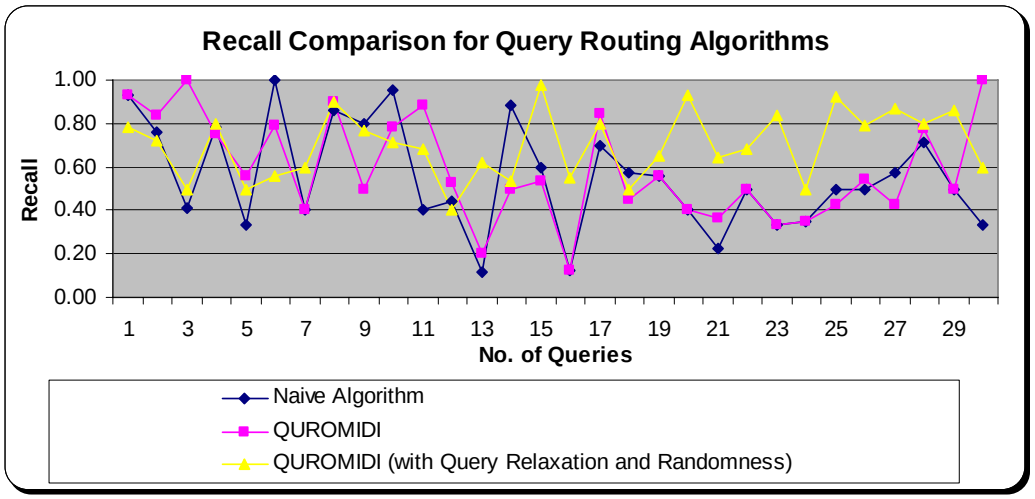



**Figure 33: This figure shows recall comparison of the three different query routing algorithms on a network of 40 agents with 30 queries tested on each**

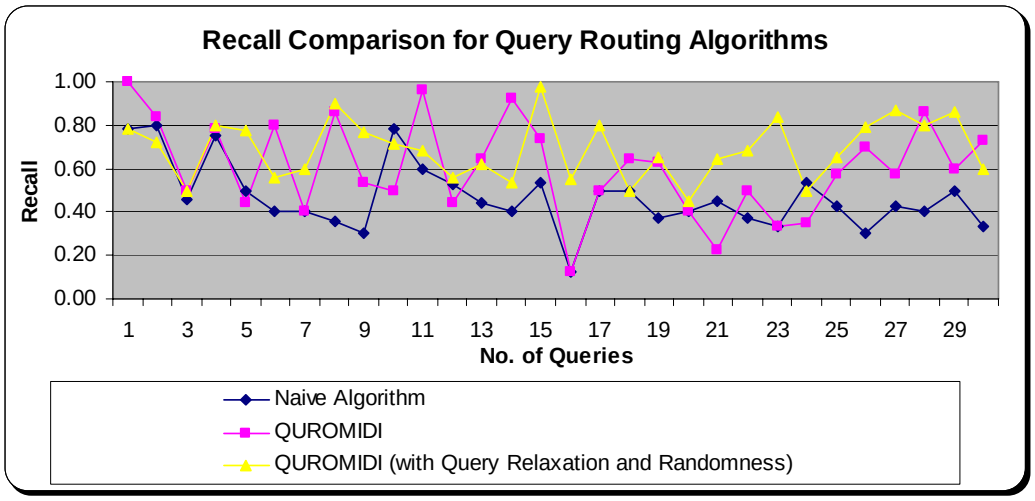



**Figure 34: This figure shows recall comparison of the three different query routing algorithms on a network of 60 agents with 30 queries tested on each**

| ***Algorithms*** | ***20 Agents*** | ***40 Agents*** | ***60 Agents*** |
|---|---|---|---|
| *Naïve* | 59% | 55% | 47% |
| *QUROMIDI* | 62% | 60% | 59% |
| *QUROMIDI (with query relaxation and randomness)* | 71% | 70% | 69% |

**Table 7: This table shows the summary of the results shown in the above charts in the form of average recall for three different query routing algorithms with different size of agent networks**

Figure 32, 33, and 34 and the data in Table 7 demonstrate that QUROMIDI with *query relaxation* and use of some *randomness* in agent selection outperforms the naïve and the simple QUROMIDI with a significant margin. Results show that with query relaxation QUROMIDI performs well not only in the same size of networks but also in the scaled up networks. The average recall rate of QUROMIDI with query relaxation on three networks of 20, 40, and 60 agents respectively is 70%. By contrast simple QUROMIDI' average recall rate on the scaled up networks is 60% and the naïve has an average recall rate of 53% percent.

In this way, with query relaxation QUROMIDI performs 10% better than the simple QUROMIDI and 17% better than the naïve one on scaled up networks. If we compare the results in same size of networks then again it performs well with a clear margin than simple QUROMIDI and the naïve algorithm. When network size is 20, it performs 12% better than the naïve and 9% better than the simple QUROMIDI. When network size is 40 it performs 15% better than the naïve and 10% better than the simple QUROMIDI and finally when network size is 60 it performs 22% better than the naïve and 10% better than the simple QUROMIDI.

### 6.4.5. Effects of Hop Count on Recall

In this section we discuss and evaluate the results to see the effect of hop count on recall. Deciding maximum hop limit is a critical issue in query routing algorithms; on one side it guarantees improved recall but on other hand it increases network load from excessive messages. So there is a trade-off between the rate of recall and the number of messages generated per query. But after observing the results, it appears that hop count matters up to a point but after that it makes no difference in the recall. We tested the network on a value of 0 to 5 for the hop count. The following are the results:

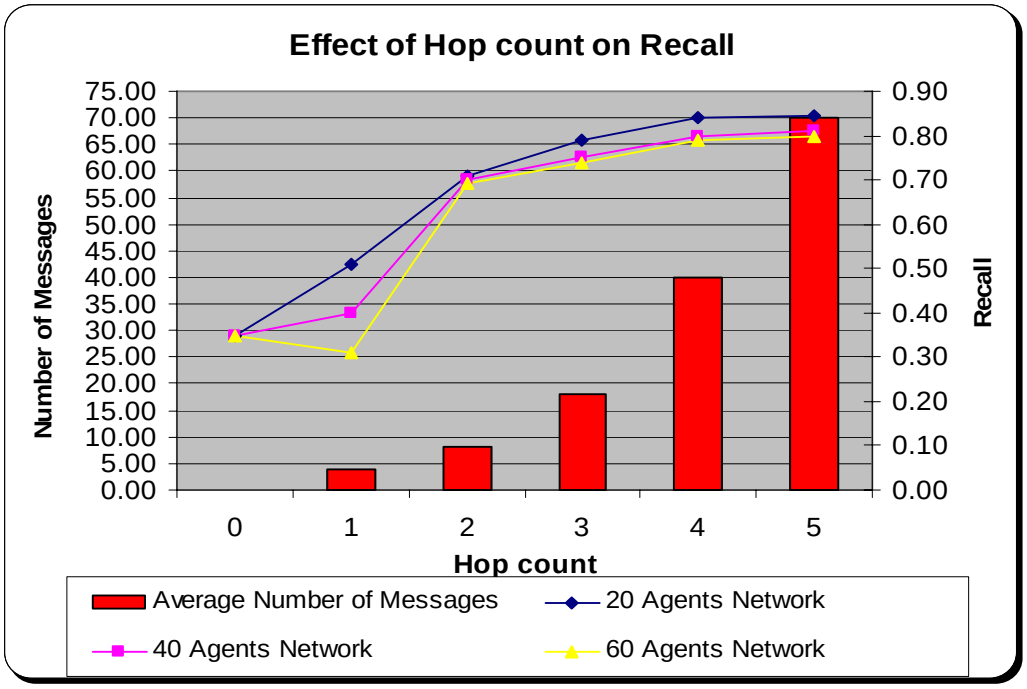



**Figure 35: This figure shows the effect on recall due to increment in hop count, tested on networks of 20, 40, and 60 agents with 30 queries on each, using intelligent algorithm with query relaxation option**

| *Network Size* | *Recall for 2 Hops* | *Recall for 5 Hops* |
|---|---|---|
| 20 Agents Network | 71% | 84% |
| 40 Agents Network | 70% | 81% |
| 60 Agents Network | 69% | 80% |

**Table 8: This table shows the increment in recall by increasing the hop count, it has been tested on intelligent networks for network size of 20, 40, and 60 agents using QUROMIDI with query relaxation option**

***Summary:*** The chart in Figure 35 and the data in Table 8 show that there is a significant improvement in recall approximately of 10%. But the alarming thing to note is the number of messages increasing dramatically. The number of messages increased at the rate of 88% from an average of 8 messages per query on a hop count of 2 to 70 messages per query on a hop count of 5. It shows that the 10% (on average) improvement in recall rate by increasing the hop count from 2 to 5 on the cost of 88% increment in number of messages is not sensible.

***Critical Analysis:*** Larger hop count could be beneficial if the number of agents with similar expertise is also increased (not just by increasing the network size). The basic idea of hop count is to set the limit on forwarding the query to other agents as well as to enable the agents to access other agents in the network which are most likely to answer the queries, in a controlled way. In our simulation, we have four agents with similar expertise which are known to each agent. As the network learns and adapts the dynamic semantic topology it forms semantic clusters of agents sharing similar expertise instead of those agents which are assigned initially (without considering the similarity in their expertise) (see section 5.5.3 for details).

Now agents know (after a certain period of learning) the most suitable agents to forward the queries to and they can access them with the already set hop count limit. Increment in the hop count limit would really not make any improvement in the recall rate instead of flooding the network with unnecessary query messages unless the number of known agents is not increased as well. Because in that case, agents would need a larger hop count limit to access other agents in the network.

# Chapter 7

# Comparison

This chapter describes the comparison between the two agent paradigms mentioned briefly in the introduction: software agents who have intelligence in themselves versus protocol-based agents who are not intelligent themselves but use intelligent protocols for intelligent behaviour. As we have already discussed software agents in detail in chapter 2. We now briefly discuss the protocol-based agents. However, for a detailed description, a reader should refer to the thesis of the other member of this joint research study [34].

## 7.1. Protocol-Based Agents

As agent based programming paradigm is evolving, more and more complex agents are being designed and developed. The need for effective coordination among agents is also growing. They need to provide support for a wide variety of interaction between businesses and individuals over the Internet. Flexibility, robustness, and extensibility are critical issues for multi-agent applications to be reliable in open systems. Agent communication protocols provide a useful framework for conversation among agents. These protocols consist of agent definitions and the actions agents are required to do. (see details in the section below)

Distributed dialogue protocols are based upon the Calculus of Communicating System (CCS). They ensure that the agents are truly autonomous and there is no centralised agent involved for coordination among agents. Two languages have been developed on the basis of distributed dialogue protocols Lightweight Coordination Calculus (LCC) and Multi Agent Protocols (MAP) [18]. LCC has been used as the protocol language for this comparison study. We will discuss LCC briefly in the next section. For details about MAP reader can refer to [19].

## 7.2. Lightweight Coordination Calculus

Lightweight Coordination Calculus (LCC) is a coordination language between agents in a multi-agent system. “The most basic behaviours are sending a message and receiving it, where sending a message may be conditional on satisfying a constraint and receiving the message may constraints on the agent accepting it” [18].

Agents preserve their autonomy property in LCC as there is no centralized agent involved between two agents for coordination. Lightweight formal methods have been used to define LCC as shown below [18]:

Framework := {Clause,....}
Clause := Agent :: Def
Agent := a(Type, Id)

Def := Agent | Message | M => Agent -> C | M <= Agent | C <- M <= Agent
Message := M => Agent | M => Agent <- C | M <= Agent | C <- M <= Agent

C:= Term | C ^ C | C v C
Type := Term
Id := Constant
M := Term

An agent A, is defined by a term A::D, where D describes the messages it is allowed to send. Different operators can be used to construct D. Constraints associated with messages should be satisfied before sending or receiving a message named as proactive and reactive constraints respectively. Complex behaviours can be specified using the connectives then, or and par which denotes sequence, choice and parallelization respectively. With the help of constraints on messages, an agent can interact according to given social norms while maintaining as much as possible of their autonomy [18].

A simple example of a scenario is modelled in the LCC Language below, it describes the interactions of two agents, agent A in the role of *requester* given a medicine

name(M) asks the agent P in the role of *pharmist* to look up the price of the given medicine M. If the *pharmist* knows the price of the medicine M then it will send an answer message to the *requester* which contains the price Y of medicine M.

a(requester(M),A) :: ask (price(M)) => a (pharmist,P) then

answer (price(Y)) <= a (pharmist,P)

a(pharmist,P) :: ask (price(X)) <= a (requester(M),A) then

answer (price(Y)) => a (requester(M),A) <-- knowsPrice (M)

In the above protocol => denotes message sending, <= denotes the message receiving and <-- defines the constraints.

## 7.3. Intelligent Agents versus Intelligent Protocols

In this section we discuss the issues which have been compared and a brief discussion of the system developed in the other part of this joint study to give an idea to the user. For details a reader should refer to [34]

### 7.3.1. A Brief Overview of Intelligent Protocol System

- When an agent joins a network, it will send advertisement messages to all of its connected agents (which is set to 4) in the network. The advertisement message consists of description of expertise of the sender. In this way, every agent sends advertisement message to all the agents it knows and receive an advertisement message from each one of them. Agents use this information when they forward a query to other agents in determining appropriate agents for the query.

- First an agent is selected on the basis of closest area identified by the user in the ACM topic tree. This first agent will receive keywords entered by the user. This agent will perform a local search its knowledge base and retrieve all the

results (if any), it finds relevant to the search.

- The querying agent will select the best agents based upon the advertisements of its known agents and forward the query to those agents (intermediate agents). These intermediate agents perform the same steps and forward the query to their known agents. This process continues until the hop count reaches its maximum value which is set to 2.

- The answering agent (the agent which initiates the answer message) also sends its expertise in the answer message to the querying agent. After receiving the answer message, the querying agent will store the expertise in its local knowledge base and use this information when it receives same or similar query next time to forward the query to appropriate agent.

### 7.3.2. Recall Comparison

Recall is a standard information retrieval measure. It determines how many documents have been returned out of all the relevant documents. As we mentioned earlier that recall is the most important factor to be considered in such decentralised environments. We have compared the overall recall rates achieved by both of these approaches to assess their effectiveness in resource recovery. Figure 36 shows the results:

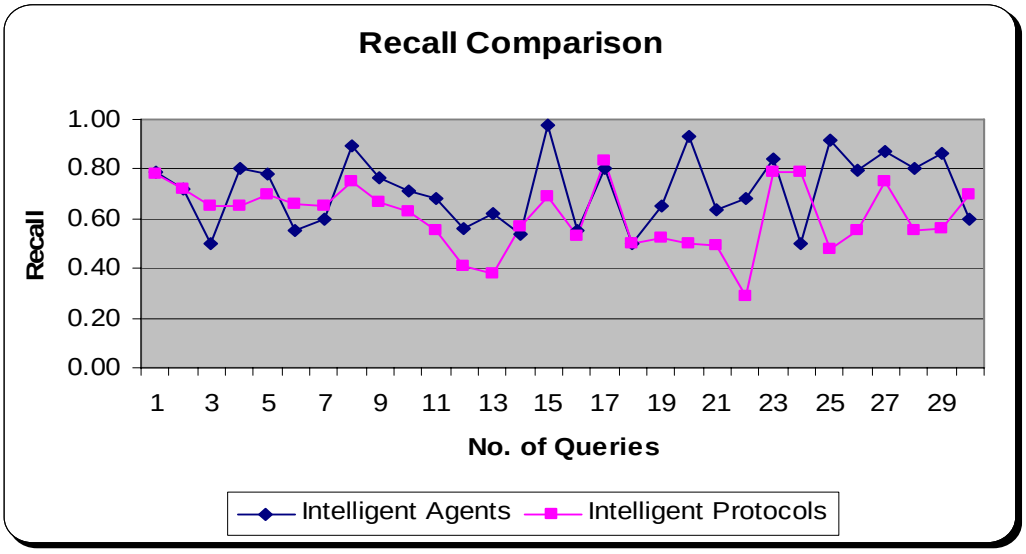



**Figure 36: Recall comparison for Intelligent Agents and Intelligent Protocols. On a network of 20 agents with 30 queries tested.**

According to these results, intelligent agents achieve an overall recall rate of 71% while agents with intelligent protocols achieve an overall recall rate of 64%. So, intelligent agents perform better than agents with intelligent protocols by having an overall lead of 7%. The reason why agents with intelligent protocols cannot perform equal or better than intelligent agents is that they do not store information acquired by network observation for each relevant query. The root cause for not storing the observed information is that these protocols have to keep themselves lightweight because if they store all this information then it would again become a complex task. There is a performance trade off in being lightweight to support flexibility, robustness, and extensibility. We will discuss the reasons in detail in the conclusion chapter (see section 8.2 for details).

# Chapter 8

# Conclusion

This chapter describes the conclusions drawn from the evaluation and comparison chapters along with some discussion.

## 8.1. Evaluation Conclusion and Discussion

The principle of self-organisation has been under discussion for a long time as a paradigm for introducing order into complex decentralised distributed systems. Specifically, the idea of self-organisation is very appealing for resource critical applications like agent-based P2P systems [7]. Self-organising systems are able to dynamically adjust with the changing environment without any external support. These systems gain performance boost over time in a non-linear fashion, until they reach a certain threshold point and meets the standard requirements. That is why such systems are called self-organising. Evaluation results show that QUROMIDI (with query relaxation and randomness) mimics the same behaviour as of self-organising systems by improving its resource discovery performance over time. It helped to justify the hypotheses made in section 6.4:

- The QUROMIDI outperforms the naïve base line algorithm in terms of recall.
- Use of the ACM topic hierarchy as an ontology and semantic similarity function helps to identify most suitable agents parsimoniously.
- Agents adapt the dynamic topology quickly and adjust accordingly, forming semantic clusters sharing similar expertise in the network.
- These semantic clusters help to increase the recall and decrease the number of messages over time.
- Use of net-workers to relax the query can enhance the performance of QUROMIDI which outperforms the simple QUROMIDI as well.
- Some randomness in agent selection helps to avoid the problem of over fitting.

## 8.2. Comparison Conclusions and Discussion

Intelligent agents perform better than intelligent protocols by 7% but still it is not straightforward to say that one paradigm outperforms the other. Each of them has its own advantages and disadvantages.

Protocol-based agents are good for an environment where features like flexibility, robustness, and extensibility of interactions are of critical importance (e-commerce or business to business applications etc.). This paradigm is helpful where one knows little about the details of agents but there is an overriding need for predictable interactions.

On the other hand, software agents, which have intelligence in themselves, are suitable for domain specific problems i.e. scientific or mission/decision critical tasks. In such domains we know what the environment is, which the interacting bodies are and how it could change. It is better to have intelligent agents in such domains to gain maximum performance.

The point of balance between these two paradigms is how much sophistication should be within the agent and how much should be in communication protocols. Protocol-based proponents believe that the communication protocols should be sophisticated because they have to interact across the domains and there are fair chances of encounter with various types of agents having different properties. If they can understand a common protocol language they can communicate effectively to achieve an overall goal using their own internal mechanisms. Their internal mechanisms (i.e. search etc.) would definitely be different from others but it does not make any difference from the point of view of interactions as long as goal is being achieved successfully. The practical trade off of this approach is that to give flexibility, robustness, and extensibility it has to compromise on performance because to keep protocols lightweight they cannot overwhelm the protocols with the information to gain maximum intelligent behaviour.

On the other hand, intelligent agents are considered suitable for task-specific, well aware domains where critical decision power is needed instead of features like

flexibility, robustness, and extensibility. This approach has to compromise on these issues to gain the maximum sophisticated behaviour capability. In this case, agents cannot interact easily with the agents in other domains because knowledge of what communications mean in the context of interaction is encoded privately.

To conclude, we can say that both of these paradigms have their distinct importance. The choice must be made carefully according to the domain requirements.

# Chapter 9

# Future Work

This chapter describes the shortcomings and improvements identified and suggested for this thesis work respectively.

## 9.1. New Directions

In this thesis report, the design principle has been to dynamically adapt the semantic topology, based on the experiences learned from successful or semantically similar queries. Experimental results show that our approach outperforms the basic approaches of random and exact matched based agent selection. However, there is still room for improvements which we will discuss in the following sections:

### 9.1.1. Organisational Model

In our simulation, we have used a simple expertise based model representing a bibliographic scenario. There are many other possibilities where more complex organisational models would have to be handled. For example in ecommerce or business-to-business applications. We have also used a simple similarity based ranking which could be made more complex to achieve better results by not only measuring the similarity in expertise but also by measuring how much information an agent contains about a particular topic.

### 9.1.2. Search Algorithms

In current system, we did not focus on efficient search mechanism with the assumption that our user will enter right and precise queries. But the precision could be enhanced by implementing efficient searching mechanisms in complex domains.

### 9.1.3. Physical Network Issues

We did not address the issues related to underlying physical network topologies. Our main focus has been on investigation of semantic topologies. It would be an interesting idea to explore the results of the deployment of semantic topology approaches on physical network topologies. Like JXTA platform could be used to test the results of discovery and semantic query routing, by our approach, in extended networks.

### 9.1.4. Field Experiment

The results presented here are simulation based. However, it would also be an interesting idea to test the model in real environment with larger networks and larger size of real knowledge.

# References

**[1]** Yu, B., Singh, M.P.: A social mechanism of reputation management in electronic commerce. In: CIA 2000. (2000) 154–165

**[2]** How Big Is The Internet? http://metamend.com/internet-growth.html

**[3]** Koubarakis 2003. MultiAgent Systems and PeertoPeer Computing: Methods, Systems, and Challenges. Invited Talk in 7th Int. Workshop on Coop. Information Agents, Finland.

**[4]** Ding, H., Solvberg, I. T., and Lin, Y. 2004. A Vision on Semantic Retrieval in P2P Network. In *Proceedings of the 18th international Conference on Advanced information Networking and Applications - Volume 2* (March 29 - 31, 2004). AINA. IEEE Computer Society, Washington, DC, 177

**[5]** Despotovic, Z., Aberer, K.: Trust -Aware Delivery of Composite Goods. *International Workshop on Agents and Peer-To-Peer Computing*, 2002.

.

**[6]** Faratin, P., N. R. Jennings, P. Buckle, and C. Sierra. (2000). Autonomated Negotiation for Provisioning Virtual Private Networks Using FIPA-Complaint agents, in Proceedings of (PAAM-2000). Manchester, UK, 185 202.

**[7]** Tempich, C., Staab, S., and Wranik, A. 2004. Remindin': semantic query routing in peer-to-peer networks based on social metaphors. In Proceedings of the 13th international Conference on World Wide Web (New York, NY, USA, May 17 - 20, 2004). WWW '04. ACM Press, New York, NY, 640-649. DOI= http://doi.acm.org/10.1145/988672.988759

**[8]** Li Y., Bandar Z, and McLean D. An approach for measuring semantic similarity between words using multiple information sources. IEEE Trans. on Knowledge and Data Eng., 15(4):871--882, 2003.

**[9]** Haase, P.; Siebes, R.; and van Harmelen, F. 2004. Peer selection in peer-to-peer networks with semantic topologies. In International Conference on Semantics of a Networked World: Semantics for Grid Databases, 2004, Paris.

**[10]** How to Classify Works Using ACM? Computing Classification System http://acm.org/class/how_to_use.html

**[11]** Aha, W. 1998. "Feature weighting for lazy learning algorithms." Feature Extraction, Construction and Selection: a Data Mining Perspective, edited by H. Liu and H. Motoda. Norwell, MA: Kluwer.

**[12]** Aha, D. W. (1997) Editorial, Artificial Intelligence Review, Volume 11, Issue 1 - 5, Feb 1997, Page 7 10.

**[13]** Panti, M., Penserini, L., and Spalazzi, L. A Multi-Agent System based on the P2P model to Information Integration.In: Autonomous Agents and Multi-Agent Systems (AAMAS 2002). (Bologna, Italy. July 16, 2002).

**[14]** Tom Mitchell. Machine Learning. McGraw Hill, 1996

**[15]** Resnik, P (1995). Using information content to evaluate semantic similarity in a taxonomy. , Proceedings of the 14th International Joint Conference on Artificial Intelligence pp 448-453.

**[16]** Berners-Lee, T., Hendler, J. and Lassila, O. The semantic Web. Scientific American (May 2001), 28-37.

**[17]** Ehrig, M., P. Haase, et al. 2003. "The SWAP Data and Metadata Model for Semantics-Based Peer-to-Peer Systems." Lecture Notes in Computer Science 2831: 144-155.

**[18]** Robertson D. 2002 Distributed Agent Dialogues. Edinburgh University.

**[19]** Walton, C. (2004) *Multi-Agent Dialogue Protocols*. Proceedings of the 8th International Symposium on Artificial Intelligence and Mathematics

**[20]** Steels, L. 1990. Cooperation between distributed agents through selforganization. In *Decentralized A.I.*, Y. Demazeau and J.-P. Muller
(Eds.), Elsevier Science.

**[21]** Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., and Schmidt, R. 2003. P-Grid: a self-organizing structured P2P system. *SIGMOD Rec.* 32, 3 (Sep. 2003), 29-33. DOI=
http://doi.acm.org/10.1145/945721.945729

**[22]** Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A. Huberman. Search in power-law networks. *Physical Review E*, 64(46135), 2001.

**[23]** Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., and Schmidt, R. 2003. P-Grid: a self-organizing structured P2P system. *SIGMOD Rec.* 32, 3 (Sep. 2003), 29-33. DOI=
http://doi.acm.org/10.1145/945721.945729

**[24]** Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., and Risch, T. 2002. EDUTELLA: a P2P networking infrastructure based on RDF. In *Proceedings of the 11th international Conference on World Wide Web* (Honolulu, Hawaii, USA, May 07 - 11, 2002). WWW '02. ACM Press, New York, NY, 604-615. DOI= http://doi.acm.org/10.1145/511446.511525.

**[25]** Ehrig, M., P. Haase, et al. 2003. "The SWAP Data and Metadata Model for Semantics-Based Peer-to-Peer Systems." Lecture Notes in Computer Science 2831: 144-155.

**[26]** Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I., and Löser, A. 2003. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In *Proceedings of the 12th international Conference on World*

*Wide Web* (Budapest, Hungary, May 20 - 24, 2003). WWW '03. ACM Press, New York, NY, 536-543. DOI= http://doi.acm.org/10.1145/775152.775229

**[27]** Balke, W., Nejdl, W., Siberski, W., and Thaden, U. 2005. Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks. In *Proceedings of the 21st international Conference on Data Engineering (Icde'05) - Volume 00* (April 05 - 08, 2005). ICDE. IEEE Computer Society, Washington, DC, 174-185. DOI= http://dx.doi.org/10.1109/ICDE.2005.115

**[28]** Kautz, H., Selman, B., and Shah, M. 1997. Referral Web: combining social networks and collaborative filtering. *Commun. ACM* 40, 3 (Mar. 1997), 63-65. DOI= http://doi.acm.org/10.1145/245108.245123

**[29]** Althoff, K. and Aamodt, A. 1996. Relating case-based problem solving and learning methods to task and domain characteristics: towards an analytic framework. *AI Commun.* 9, 3 (Sep. 1996), 109-116.

**[30]** **K**rackhardt, D., Krebs, V. *Kite Network ideas* *http://www.orgnet.com/sna.html*

**[31]** Morville, P. *Social Network Analysis* http://semanticstudios.com/publications/semantics/000006.php

**[32]** Ehrig, M., Schmitz, C., Staab, S., Tane, J., Tempich, C. 2003.: Towards evaluation of peer-to-peer-based distributed knowledge management systems. In: Proceedings of the AAAI Spring Symposium "Agent-Mediated Knowledge Management (AMKM-2003)".

**[33]** *Association of Computing Machinary* http://www.acm.org

**[34]** Pervez Z. 2005. Semantic Query Routing in agent-based P2P systems using LCC Protocols. Thesis (MSC). University of Edinburgh

**[35]** Melville, L., Walkerdine, J., Sommerville, I.: Ensuring dependability of P2P applications at architectural level. Technical Report IST-2001-32708, Lancaster University, http://polo.lancs.ac.uk/p2p/Documents/PropertiesDeliverable.pdf (2002)

**[36]** Barkai, D. 2001 *Peer-to-Peer Computing: Technologies for Sharing and Collaborating on the Net*. Intel Press

**[37]** Yang, B., Garcia-Molina, H. Comparing hybrid peer-to-peer systems. In *VLDB*, pages 561–570, 2001.

**[38]** Androutsellis-Theotokis, S. and Spinellis, D. 2004. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.* 36, 4 (Dec. 2004), 335-371. DOI= http://doi.acm.org/10.1145/1041680.1041681.

**[39]** Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. 2001. 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* (San Diego, California, United States). SIGCOMM '01. ACM Press, New York, NY, 149-160. DOI= http://doi.acm.org/10.1145/383059.383071

**[40]** Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. 2001. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications* (San Diego, California, United States). SIGCOMM '01. ACM Press, New York, NY, 161-172. DOI= http://doi.acm.org/10.1145/383059.383072

**[41]** Rowstron, A., and Druschel, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM Int'l Conf. on Distributed Systems Platforms* (Nov. 2001)

**[42]** Gedye, D., Kasnoff, C.. SETI@home project.

http://setiathome.ssl.berkeley.edu/project.html.

**[43]** Napster, I.: Napster. http://www.napster.com

**[44]** Gnutella: (Development home page) http://gnutella.wego.com/.

**[45]** Russell, S. J. and Norvig, P. 2003 Artificial Intelligence: A Modern Approach. 2. Pearson Education.

**[46]** Yolum P, Singh MP(2002)''Agent-Based Approach for Trustworthy Service Location''.Proceedings of the Workshop on Agents and Peer-to-Peer Computing (AP2PC 2002) at AAMAS2002, Bolgona, Italy.