

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
MOHAMED BOUDIAF UNIVERSITY - M'SILA

FACULTY: Mathematics and Informatics

DEPARTMENT of Computer Science

N°:.....



DOMAIN: Mathematics and Informatics

FIELD: Computer Science

SUB-FIELD: RTIC

**Dissertation submitted in partial fulfillment of the requirements for
the degree of MASTER
By: ZOUAOUI HAMZA**

SUBJECT

**Intelligent mechanism for communication in
P2P networks**

Publicly defended on: 23/06/2018 before Jury composed of:

Ms. Chalabi baya

M'sila University

Chairman

Mr. MOHAMED SAHRAOUI

M'sila University

Supervisor

Mr. Gasmia salah

M'sila University

Examiner

Academic Year: 2017/2018

Acknowledgements

First of all, praise be to Allah, I would like to thank Mr. Sahraoui Mohamed my supervisor for this support and guidance through this work. I would like also to thank all the professors of RTIC for their valuable advices and encouragement.

Finally, I am very grateful to my parents and my family for their unlimited love and support during my studies. This work is dedicated to them and to all my friends who have supported me.

Table of Contents

Table of Contents.....	I
List of figures.....	III
List of tables.....	IV
General introduction.....	1
Chapter 1 generality about client/server and Peer-to-peer networks.....	2
1. Introduction	3
2. Client/Server Networks.....	3
2.1 Definitions of Client/Server.....	3
2.2 Client-Server architectures.....	3
2.3 Advantages.....	4
2.4 Disadvantages.....	4
3. Peer to peer networks.....	5
3.1 Peer to Peer definitions.....	5
3.2 Characteristics of the Peer-To-Peer Networks.....	6
3.3 Peer-To-Peer Architectures.....	6
3.3.1 Centralized P2P Systems.....	7
3.3.2 Decentralized P2P Systems.....	8
3.3.2.1 The network structure.....	9
3.3.2.2 The logical network topology.....	9
3.3.3 Hybrid P2P Systems.....	10
4. Peer-to-Peer Networks Vs Client/Server Networks:	11
8. Conclusion.....	12
 Chapter 2 searching techniques in unstructured peer to peer networks.....	 13
1. Introduction.....	14
2. Searching Techniques in unstructured Peer-to-Peer Networks.....	14
2.1. Blind searches.....	14
2.1.1. Gnutella.....	15
2.1.2. Iterative Deepening.....	15
2.1.3 k -walker random walk and related schemes.....	17
2.1.4 Modified-BFS.....	18
2.2. Informed Search.....	19

2.2.1 Reinforcement Learning.....	19
2.2.1.1 Intelligent-BFS.....	19
2.2.1.2 Adaptive Probabilistic Search (APS).....	20
2.2.1.3 Distributed Resource Location Protocol (DRLP).....	21
2.2.1.4 Intelligent Search:.....	22
3. Conclusion:.....	22
 Chapter 3 Design and Implementation.....	23
1. Introduction.....	24
2. Proposed solution.....	24
2.1. General explanation.....	24
2.2. Routing table	24
2.3. Diagrams	26
2.3.1. Use case diagram	26
2.3.2. Sequence diagram	27
2.4. Algorithm	28
3. Implementation	30
3.1. Simulator PeerSim.....	31
3.2. Implementation of the programme	32
3.3. Simulation results	37
4. Comparison:	41
5. Conclusion:.....	41
General Conclusion.....	42

List of figures:

Figure1.1 Flat architecture (left); Hierarchical architecture (right).....	4
Figure 1.2 Peer-to-Peer Architecture.	6
Figure 1.3 Centralized Peer-to-Peer Architecture.....	7
Figure 1.4 Decentralized Peer-to-Peer Architecture.....	8
Figure 1.5 Hybrid Peer-to-Peer Architecture	11
Figure 2. 1Gnutella network.....	15
Figure 2. 2Iterative deepening example.....	17
Figure 2. 3 Modified-BFS example.....	18
Figure 2. 4 Adaptive Probabilistic Search example.....	21
Figure 3. 1 RTIDS example case (01).....	25
Figure 3. 2 RTIDS example case (02).....	25
Figure 3. 3 use case diagram	26
Figure 3. 4 Sequence diagram of search technique.....	27
Figure 3. 5 Sequence diagram of Messages exchanged.	28
Figure 3.6 The time it takes to get a response (test 1).....	37
Figure 3.7 The time it takes to get a response (test 2).....	38
Figure 3.8 Number of messages sent in order to receive a response (test 1).....	38
Figure 3.9 Number of messages sent in order to receive a response (test 2).....	39
Figure 3.10 the depth necessary to reach the result.....	40

List of tables:

Table 1. 1 Peer-to-Peer Networks Vs Client/Server Networks.....	12
Table 3.1 comparison between the RLIDS and IDS.....	41

General introduction

The advances in public networks and the deployment of powerful personal computing units by end users has brought a shift from the traditional Client- Server computing model to the Peer-to-Peer computing.

Peer-to-Peer (P2P) systems are distributed systems in which nodes of equal roles and capabilities exchange information and services directly with each other. In recent years, P2P has emerged as a popular way to share huge volumes of data, Nodes in a P2P network normally play equal roles, and therefore, these nodes are also called peers, Peer-to-Peer networks have become a major research topic over the last few years. Object location is a major part in the operation of these distributed systems.

Today, the most popular P2P applications operate on unstructured networks. In these networks, peers connect in an ad-hoc fashion, the location of the documents is not controlled by the system and no guarantees for the success of a search are offered to the users, when a peer wants to find a desired piece of data in this type of P2P networks, the search query must be flooded through the network to find as many peers as possible that share the data. Flooding causes a very high amount of signaling traffic in the network, for this reason, many of searching schemes have been proposed to improve the information research mechanism. We propose in this study a hybrid scheme between adaptive learning (reinforcement learning) and iterative deepening to improve this research mechanism. Through the well results that shows the good improvement of the deepening method by the integrating of adapting mechanism, we motivated to present our work that is organized on three chapters; in the first we overview the peer to peer networks, in the second we review the different research techniques used in unstructured peer to peer networks, and finally we explain our solution design and its simulation results.

CHAPITRE 1

GENERALITY ABOUT PEER-TO-PEER NETWORKS

1. Introduction:

In this chapter, we present some basic knowledge about client/server and P2P networks including its definition, the layer architecture, examples on applications of P2P and compare them.

2. Client/Server Networks:

2.1 Definitions of Client/Server:

- “A Client-Server network is a distributed network which consists of one higher performance system, the server, and several mostly lower performance systems, the clients. The server is the central registering unit as well as the only provider of content and service. A client only requests content or the execution of services, without sharing any of its own resources.” [21]
- “A Client-Server architecture is a network architecture in which each computer or process on the network is either a client or a server. Servers are powerful computers or processes dedicated to managing disk drives (file servers), printers (print servers), or network traffic (network servers). Clients are PCs or workstations on which users run applications. Clients rely on servers for resources, such as files, devices, and even processing power.” [22]

2.2 Client-Server architectures:

Client-Server architectures can be classified into flat and hierarchical. If the Client-Server model is flat, all clients communicate only with a single server, if the Client-Server model is hierarchical the servers of one level are acting as clients to higher level servers. A pretty good example is a request of a certain web page. the user enters a URL into the web browser (client). The client establishes a connection to his nearest name server to ask for the address. If that server does not know the name, it delegates the query to the authority for that namespace. That query, in turn, may be delegated to a higher authority, all the way up to the root name servers for the Internet as a whole. Name servers operate both as clients and as servers, see **Figure1.1**.

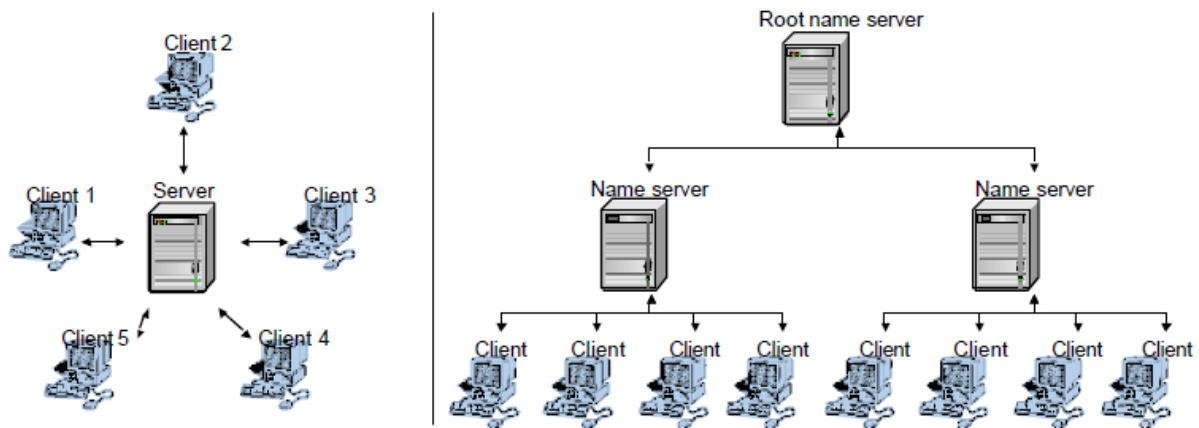


Figure1.1 Flat architecture (left); Hierarchical architecture (right)

2.3 Advantages:

The following is advantages of the Client-Server architecture:

- * Data management is much easier because the files are in one location. This allows fast backups and efficient error management. There are multiple levels of permissions, which can prevent users from doing damage to files.
- * The server hardware is designed to serve requests from clients quickly. All the data are processed on the server, and only the results are returned to the client. This reduces the amount of network traffic between the server and the client machine, improving network performance.
- * Thin client architectures allow a quick replacement of defect clients, because all data and applications are on the server.

2.4 Disadvantages:

The following is disadvantages of the Client-Server architecture:

- * Client-Server-Systems are very expensive and need a lot of maintenance.
- * The server constitutes a single point of failure. If failures on the server occur, it is possible that the system suffers heavy delay or completely breaks down, which can potentially block hundreds of clients from working with their data or their within companies high costs could accumulate due to server downtime.

3. Peer to peer networks

3.1 Peer to Peer definitions:

There has no precise definition of P2P in the literature today. Many definitions are proposed trying to capture main features of P2P systems. following are typical definitions.

Definition_N°1: “a system to be P2P if the elements that form the system share their resources in order to provide the service the system has been designed to provide. the elements in the system both provide services to other elements and request services from Other elements” [1].

Definition_N°2: “Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority” [2].

Definition_N°3: “A distributed network architecture may be called a Peer-to-Peer (P-to-P, P2P...) network, if the participants share a part of their own hardware resources (Processing power, storage capacity, network link capacity, printers ...). These shared resources are necessary to provide the Service and content offered by the network (e.g. file sharing or shared workspaces for collaboration). They are accessible by other peers directly, without passing intermediary entities. The participants of such a network are thus resource (Service and content) providers as well as resource (Service and content) requestors (Servent-concept)” [3].

Peer:

“A peer is a network node that can act as a client or a server, with or without centralized control, and with or without continuous connectivity. The term “peer” can apply to a wide range of device types, including small handheld and powerful server-class machines that are closely managed.” [23]

Node:

“A node is a computing device residing on a network. Nodes may be general-purpose computers, or they may be specialized to provide particular services or capabilities (e.g. a storage node or control node). Note the term computing device is used in the most generic sense in that a node can range from a multi processor server to embedded systems.” [23]

3.2 Characteristics of the Peer-To-Peer Networks:

The characteristics of P2P systems according to the definitions include:

- A) Shared resources: Elements or participants in P2P systems share the resources with each other in order to provide the services or content offered by these systems.
- B) Self-organized: P2P systems have ability to self-organize into network topologies. Dual role. Elements or participants in P2P systems both provide resources to others and request resources from others.
- C) Stability: P2P systems have ability to adapt with failures and accommodate with transient populations of participants or elements.
- D) Autonomy: Participants work without the support of a central servers or authority.

3.3 Peer-To-Peer Architectures:

In general, we can categorize the systems into three broad categories, centralized decentralized and hybrid.

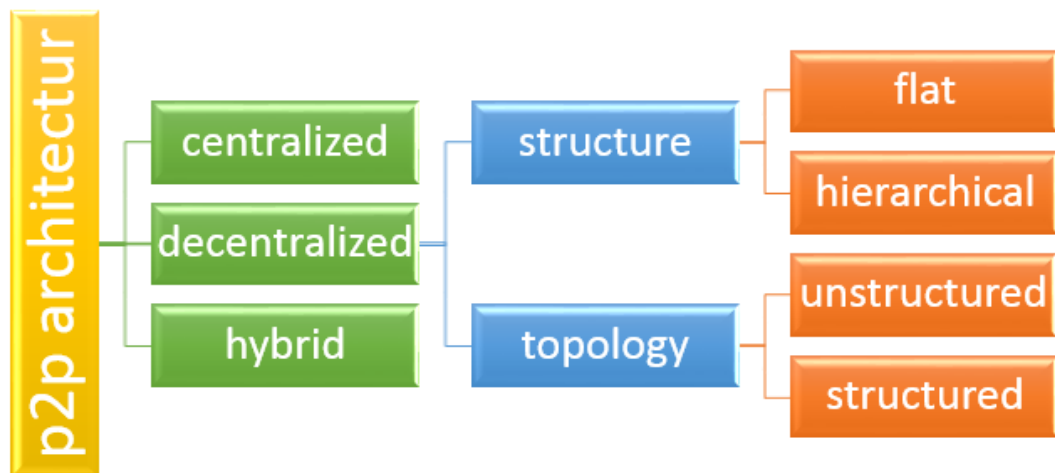


Figure 1.2 Peer-to-Peer Architecture

3.3.1 Centralized P2P Systems:

Centralized P2P systems like a client-server system, there are one or more central servers, which help peers to locate their desired resources or act as task scheduler to coordinate actions

among them. To locate resources, a peer sends messages to the central server to determine the addresses of peers that contain the desired resources (e.g., Napster [5]), or to fetch *work units* from the central server directly (e.g., BOINC [13]). However, like a decentralized system, once a peer has its information/data, it can communicate directly with other peers (without going through the server anymore). As in all centralized systems, this category of P2P systems are susceptible to malicious attacks and single point of failure. Moreover, the centralized server will become a bottleneck for a large number of peers, potentially degrading performance dramatically. Finally, this type of system lacks scalability and robustness [4].

Example: Napster

Music file sharing is perhaps one of the fastest growing applications in the Internet, and Napster certainly played a critical role in facilitating music file exchange over the net. In Napster, each user (computer) acted as a producer of content. Thus, one can view the system as a collection of MP3 files that are distributed over the personal computers of Napster users. To enable users to locate music files, Napster employed a centralized server that stored the locations of the nodes that own the files. Thus, a request will be channeled to the central server to obtain the list of owner nodes. Actual data exchange between two peers can proceed without any further intervention from the central server [5].

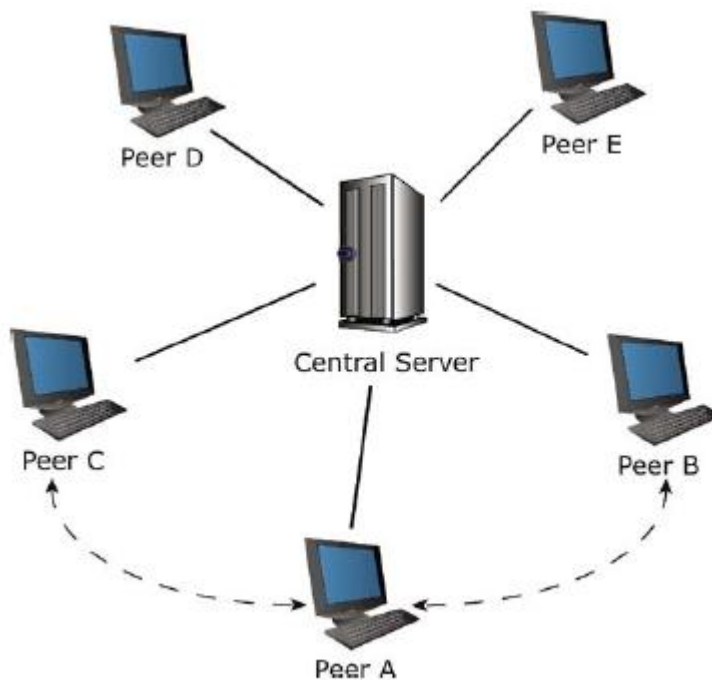


Figure 1.3 Centralized Peer-to-Peer Architecture

Fig. 1.3 The centralized P2P system. Peer A submits a request to the central server to acquire a list of nodes that satisfy the request. Once Peer A obtains the list (which contains Peer B and Peer C), it communicates directly with the nodes.

3.3.2 Decentralized P2P Systems:

In a decentralized P2P system, peers have equal rights and responsibilities. Each peer has only a partial view of the P2P network and offers data/services that may be relevant to only some queries/peers. As such, locating peers offering services/data quickly is a critical and challenging issue. The advantages of these systems are obvious:

- ✓ They are immune to single point of failure,
- ✓ Possibly enjoy high performance, scalability, robustness, and other desirable features.

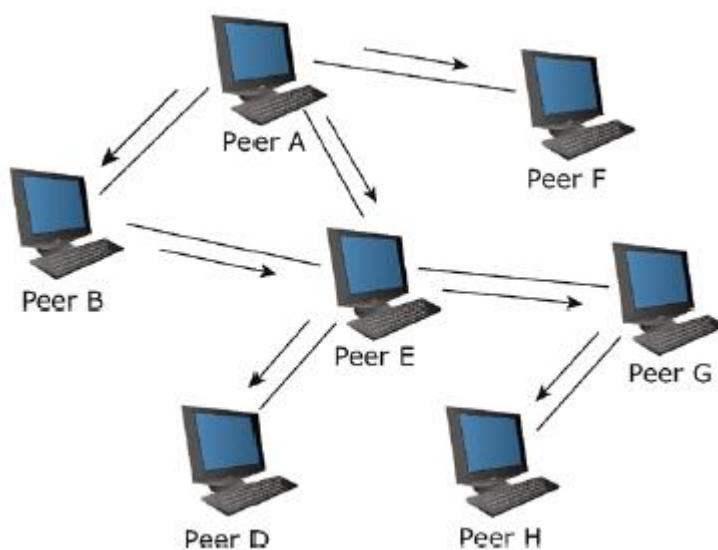


Figure 1.4 Decentralized Peer-to-Peer Architecture

Figure 1.4: Peer a requests for some data that Peer D and Peer H have. The query will be broadcasted to the neighbors of Peer A, and gradually, to the other peers in the whole network.

3.3.2.1 The network structure:

Can be flat (single-tier) or hierarchical (multi-tier). In a flat structure, the functionality and load are uniformly distributed among the participating nodes. It turns out that most of the

existing decentralized systems are nonhierarchical. On the other hand[14] hierarchical design naturally offers certain advantages including fault isolation and security, effective caching and bandwidth utilization, hierarchical storage and so on. In a hierarchical structure, there are essentially multiple layers of routing structures. For example, at a national level, there may be a routing structure to interconnect states; within each state, there may be another routing structure for universities within the state; and within each university, there may be yet another level that connects departments, and so on. Representatives of this category are the super-peer architecture [15].

3.3.2.2 The logical network topology:

Structured or unstructured. The difference between these two designs lies in how queries are being forwarded to other nodes

A) Unstructured P2P system:

. In an unstructured P2P system, each peer is responsible for its own data, and keeps track of a set of neighbors that it may forward queries to. There is no strict mapping between the identifiers of objects and those of peers. This means:

- ✓ Locating data in such a system is challenging since it is difficult to precisely predict which peers maintain the queried data.
- ✓ There is no guarantee on the completeness of answers (unless the entire network is searched).
- ✓ There is no guarantee on response time (except for the worst case where the entire network is searched).

The famous forerunners of unstructured P2P systems are FreeNet [17] and the original Gnutella. The former applies unicast-based lookup mechanisms to locate expected resources, which is inefficient in terms of response time, but efficient with respect to the bandwidth consumption and the number of messages used; the latter adopts flooding-based routing strategy, which is efficient in terms of response time but inefficient in bandwidth consumption and the number of messages used (since the network is flooded with exponential number of messages).

B) Structured P2P system:

In a structured P2P system, data placement is under the control of certain predefined strategies (generally a distributed hash table, or simply DHT).

In other words, there is a mapping between data and peers. (Very often, for security/ privacy reasons, the owners have full control over their own data. Instead, it is the metadata that is being “inserted” into the P2P network, e.g., (id, ptr) pairs that indicate that object with identifier id is located at peer pointed to by ptr. However, we shall use the term *data* in our discussion to refer to both.) More importantly, these systems provide a guarantee (precise or probabilistic) on search cost. This, however, is typically at the expense of maintaining certain additional information. Employing the principle of the mapping, most of the structured P2P systems, including CAN [18], Chord [19], and [20] Pastry, adopt the key-based routing (KBR) strategy to locate the desired resource. As a result, a request can be routed to the peer who maintains the desired data quickly and accurately. However, since the placement of data is tightly controlled, the cost of maintaining the structured topology is high, especially in a dynamic network environment, where peers may join and leave the network at will [4].

Example: Gnutella

Gnutella [6] is a large peer-to-peer network. It was the first decentralized peer-to-peer network of its kind, leading to other, later networks adopting the model. It celebrated a decade of existence on March 14, 2010 and has a user base in the millions for peer-to-peer file sharing.

In June 2005, gnutella's population was 1.81 million computers increasing to over three million nodes by January 2006. In late 2007, it was the most popular file sharing network on the Internet with an estimated market share of more than 40%. When the user wants to do a search, the client sends the request to each actively connected node. In version 0.4 of the protocol, the number of actively connected nodes for a client was quite small (around 5), so each node then forwarded the request to all its actively connected nodes, and they in turn forwarded the request, and so on, until The packet reached a predetermined number of hops from the sender (maximum 7) .

3.3.3 Hybrid P2P Systems:

The main advantage of centralized P2P systems is that they are able to provide a Quick and reliable resource locating. Their limitation, however, is that the scalability Of the systems is affected by the use of servers. While decentralized P2P systems Are better than centralized P2P systems in this aspect, they require a longer time in Resource locating. As a result, hybrid P2P systems have been introduced to take advantages Of both centralized and decentralized

architectures. Basically, to maintain the scalability, similar to decentralized P2P systems, there are no servers in hybrid P2P systems. However, peer nodes that are more powerful than others can be selected to act as servers to serve others. These nodes are often called super peers. In This way, resource locating can be done by both decentralized search techniques and Centralized search techniques (asking super peers), and hence the systems benefit from the search techniques of centralized P2P systems.

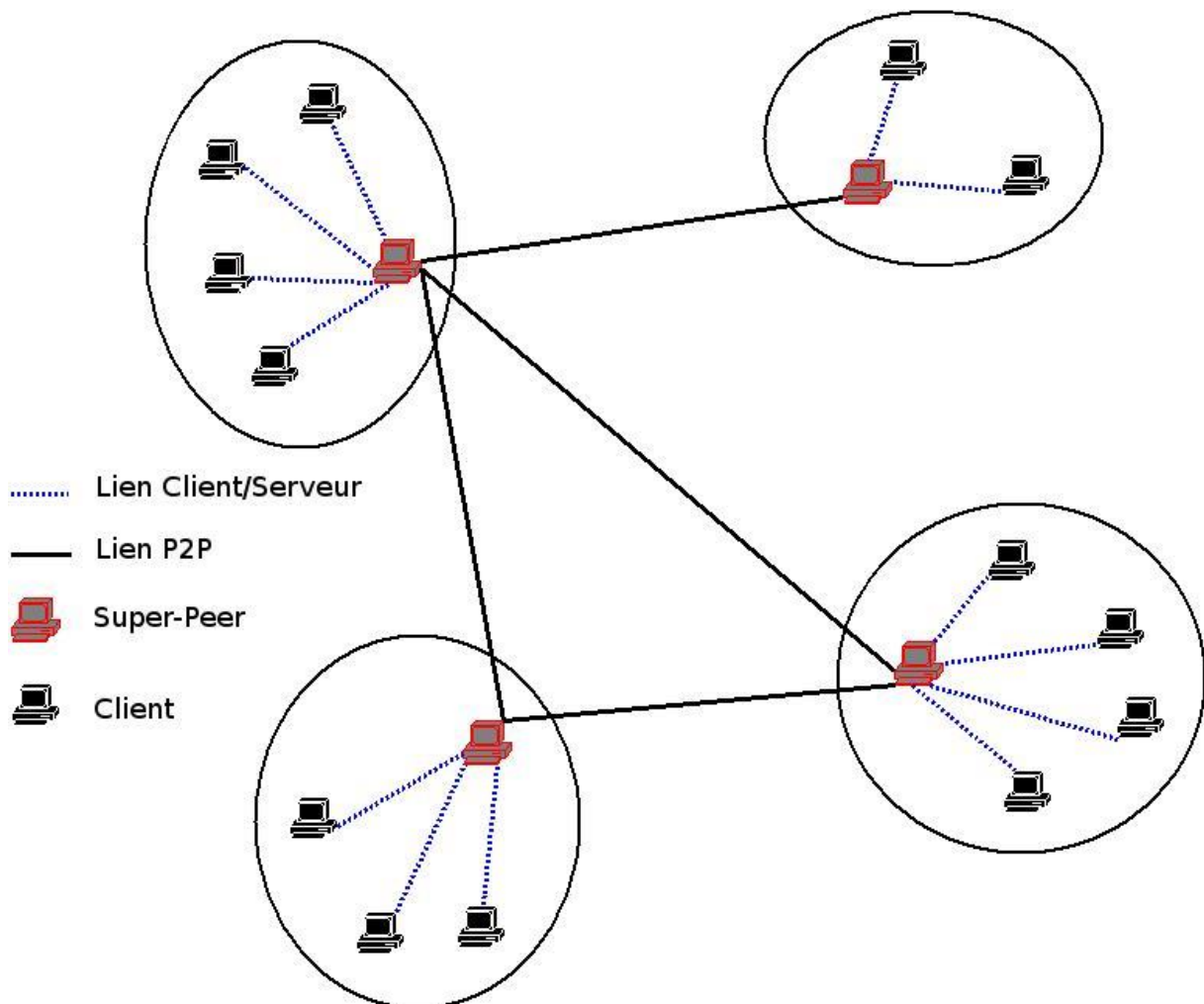


Figure 1.4 Hybrid Peer-to-Peer Architecture

4. Peer-to-Peer Networks Vs Client/Server Networks:

We evaluate the performance of a content distribution service with respect to reliability and efficiency. The considered technology for realizing such a service can either be a traditional client/server (CS) architecture or a peer-to-peer (P2P) network. In CS, the capacity of the server is the bottleneck and has to be dimensioned in such a way that all requests can be accommodated at any

time, while a P2P system does not burden a single server since the content is distributed in the network among sharing peers. However, corrupted or fake files may diminish the reliability of the P2P service due to downloading of useless contents.

Client/Server	P2P	Explanation
Managed	Self-organized	Client-server systems are centralized and need to be managed by an administrator. P2p systems are self-organized
Configured	Ad-hoc	Client-server systems need to be locally configured. P2p systems take ad-hoc decisions i.e. load balancing
Lookup	Discover	In the Client – server model we have to use specific entities which location is well – known in order to locate a resource. In the p2p model this is not always necessary (e.g. flood variations)
Hierarchy	Mesh	Client server models follow a hierarchy since clients take resources from servers. This does not happen in p2p since the organization is usually decentralized and resources may come and go from any peer leading to a formation of a mesh.
Static	Mobile	Servers must be static (e.g reside in the same IP) in order to be located by the clients. In p2p resource sharing entities may change point of presence and be also located.
Server dependencies	Independent lifetime	Servers must have extended uptime in order to provide clients with resources. In p2p systems, nodes might enter or leave the system arbitrarily.
IP-centric	Also non-IP	In p2p systems different protocols may be used in order to facilitate services such as routing.
DNS-based	Custom naming	Client – server systems uses the DNS in order to locate naming. P2P systems mainly use custom naming methods in order to cope with special system properties, such as node independent lifetimes.

Table 1.1 Peer-to-Peer Networks Vs Client/Server Networks

5. Conclusion:

After we give a general idea on important sides of the client/server networks and peer to peer networks and compare them, we will focus on the second chapter on one special type named unstructured peer to peer networks.

CHAPTER 2

SEARCHING TECHNIQUES IN UNSTRUCTURED PEER TO PEER NETWORKS

1. Introduction

In this chapter, we will study the different techniques and schemes proposed in the literature for unstructured peer to peer networks we have choose the unstructured type for P2P networks because of its speed ability of scaling, simplicity of implementing, the most popular P2P applications operate on unstructured networks and contains the intelligent searching techniques.

2. Searching Techniques in unstructured Peer-to-Peer Networks

In an unstructured P2P system, no rule exists that strictly defines where data is stored and which nodes are neighbors of each other. To find a specific data item, early work such as the original Gnutella [6] used flooding, which is the Breadth First Search (BFS) of the overlay network graph with depth limit D . D refers to the system-wide maximum TTL of a message in terms of overlay hops. In this approach, the *querying node* sends the query request to all its neighbors each neighbor processes the query and returns the result if the data is found. This neighbor then forwards the query request further to all its neighbors except the querying node. This procedure continues until the depth limit D is reached. Flooding tries to find the maximum number of results within the ring that is centered at the querying node and has the radius: D -overlay-hops However, it generates a large number of messages (many of them are duplicate messages) and does not scale well.

There are several ways to classify the searching schemes in unstructured P2P systems like regular-grained or coarse-grained, deterministic or probabilistic, BFS based or Depth First Search (DFS) based, blind search or informed search, we chose this latter [7].

2.1 Blind searches:

In a blind search, peers has no information related to file location. Therefore, other peers are randomly probed whether they have a requested file or not. Blind search protocols are simple and require peers to maintain a minimum amount of knowledge about network organization.

2.1.1 GNUTELLA:

The original Gnutella algorithm uses flooding (BFS traversal of the underlying graph) for object discovery and contacts all accessible nodes within the TTL value. Although it is simple and manages to discover the maximum number of objects in that region, the approach does not scale, producing huge overhead to large numbers of peers [6].

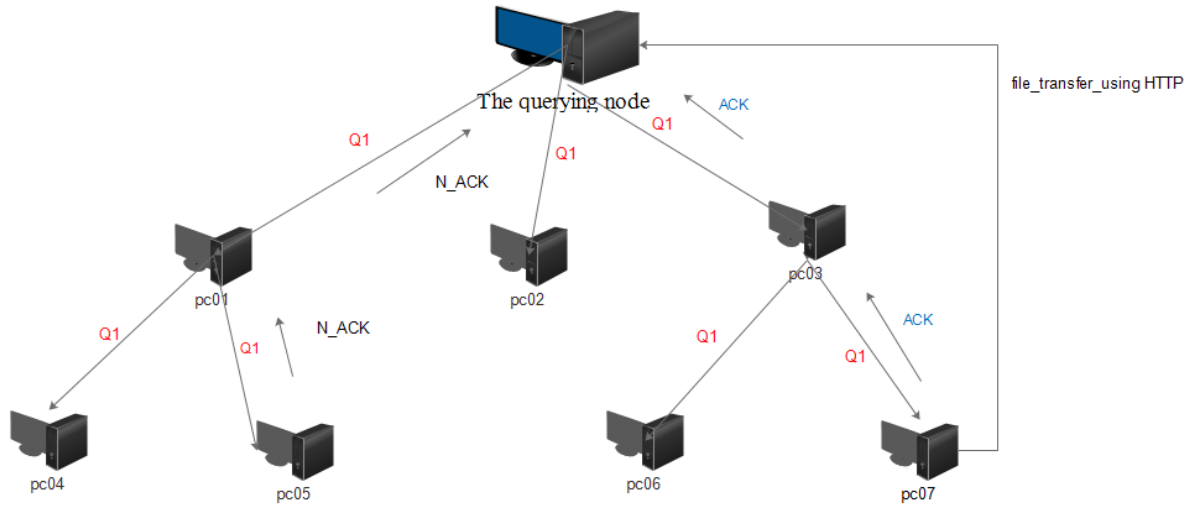


Figure 2.1 Gnutella network.

Figure 2.1: the querying node forwards the query message to all neighbors until the query arrive to node has the desired file respond by HIT_Q, then transfer the file HTTP.

2.1.2 Iterative deepening:

The idea of iterative deepening from artificial intelligence and used it in P2P searching. This method is also called *expanding ring*. In this technique, the querying node periodically issues a sequence of BFS searches with increasing depth limits $D1 < D2 < \dots < Di$. The query is terminated when the query result is satisfied or when the maximum depth limit D has been reached. In the latter case, the query result may not be satisfied. All nodes use the same sequence of depth limits called *policy P* and the same time period W between two consecutive BFS searches.

Example (1): assume that $P = \{3, 5, 8\}$, $W = 6$ seconds. The query node S first sends a BFS search with depth limit 3 to all its neighbors via a *query message*. This BFS search message will reach all nodes within 3-hops distance from S . These nodes will process this BFS message and store (*freeze*) that message for a time period ($> W$) when they receive it. If any desired data is located on these nodes, the data will be sent back to S . If the query is satisfied

within $W (= 6)$ seconds following the first BFS search, S will terminate the query and will not continue.

Otherwise, S will initiate the second BFS search with depth limit 5 via a *resend message*. The resend message carries the same query ID as in the corresponding query message. Any node within 2-hops distance from S will simply forward the resend message to all its neighbors after receiving it. The nodes at 3-hops distance from S will drop the resend message and then unfreeze the stored query message with the matching query ID. “Unfreeze” means forwarding the respective stored query message with a new depth limit 2 (? 5? 3) to all its neighbors. This unfrozen query message will be processed similarly to the query message in the first BFS search.

If the resend message with maximum depth limit 8 is sent by the querying node, nodes within 8 hops distance from S will not store (freeze) this query message. The querying node will not issue another resend message with a larger depth limit.

2.1.2.1 Advantage:

- Complete search
- Not susceptible to infinite loops.
- Small intermediate storage.

2.1.2.2 Disadvantage:

- Send lots of messages in the network
- Take a great time to reach the goal.
- Use a large portion of memory during the search.
- Re-visit multiple nodes many times with knowledge that does not contain the desired goal.

Example (2):

We have a network consisting of 15 nodes and desired goal in the depth=03, so we must visit all the nodes of network many times.

The nodes in the depth 01 we visit it 3 times, 2 times for nothing.

The nodes of the depth 02 we visit it 2 times, 1 times for nothing.

This for a network consisting of 15 nodes what for a network of 10000 nodes, it's a big lost time and many messages sent without interest **see fig 2.2.**

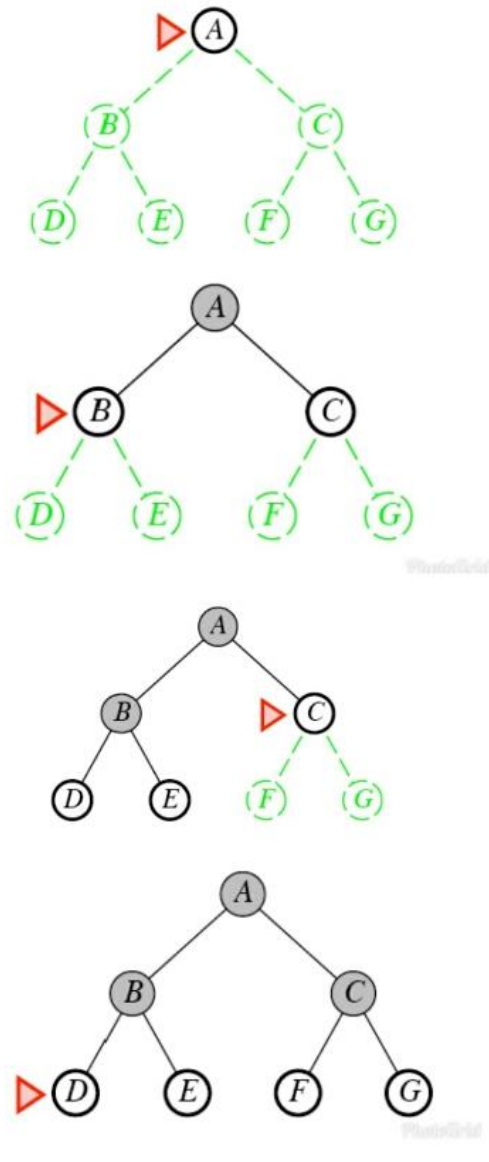


Figure 2.2 Iterative deepening example.

2.1.3 k -walker random walk and related schemes:

In the *standard random walk* [9] algorithm, the querying node forwards the query message to one randomly selected neighbor. This neighbor randomly chooses one of its neighbors and forwards the query message to that neighbor. This procedure continues until the data is found. Consider the query message as a walker. The query message is forwarded in the network the same way a walker randomly walks on the network of streets. The standard random walk algorithm uses just one walker. This can greatly reduce the message overhead but causes longer searching delay.

In the k -walker random walk algorithm [9], k walkers are deployed by the querying node. That is, the querying node forwards k copies of the query message to k randomly selected neighbors. Each query message takes its own random walk. Each walker periodically “talks” with the querying node to decide whether that walker should terminate. Nodes can also use soft states to forward different walkers for the same query to different neighbors. *k-walker random walk algorithm* attempts to reduce the routing delay. On average, the total number of nodes reached by k random walkers in H hops is the same as the number of nodes reached by one walker in kH hops. Therefore, the routing delay is expected to be k times smaller.

A similar scheme is the *two-level random walk* [7]. In this scheme, the querying node deploys $1/k$ random walkers with the TTL being $1/l$. When the TTL $1/l$ expires, each walker forges $2/k$ random walkers with the TTL being $2/l$. All nodes on the walkers’ paths process the query. Given the same number of walkers, this scheme generates less duplicate messages but has longer searching delays than the k -walker random walk.

2.1.4 Modified-BFS:

This is a variation of the flooding scheme, with peers randomly choosing only a ratio of their neighbors to forward the query to. This algorithm certainly reduces the average message production compared to the previous method, but it still contacts a large number of peers [11].

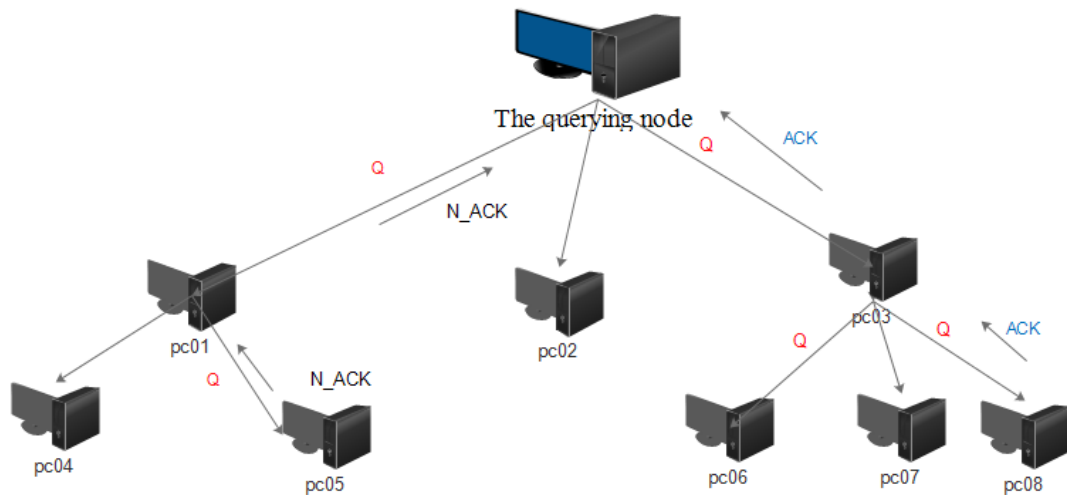


Figure 2.3 Modified-BFS example.

Figure 2.3:the querying node send the query to pc01 ,pc03 and exempted pc02 the same process in pc01 and pc03 chose a ratio of their neighbors ,pc08 has the desired so send ACK to pc03 and pc03 send ACK to the querying node .

2.2 Informed Search

In these systems, peers maintain additional information about file locations which can be useful for the search. For example, a peer may have an index of files offered by other peers connected to it. Based on this additional information, a peer decides which peers should be contacted.

2.2.1 Reinforcement Learning:

The idea of Reinforcement learning is an agent (node) learn how to behave in an environment by performing action and seeing results and profiting them, when applying RL to P2P searching, each node is a distributed learner and the P2P network is its environment, in recent years we've seen a lot of improvements in this fascinating area of research, in this part we focus on those algorithms of reinforcement learning in p2p system

2.2.1.1 Intelligent-BFS:

This is an informed version of the modified-BFS algorithm. Nodes store query-neighbor tuples for recently answered requests from (or through) their neighbors in order to rank them. First, a peer identifies all queries similar to the current one, according to a query similarity metric; it then choses to forward to a set number of its neighbors that have returned the most results for these queries. If a hit occurs, the query takes the reverse path to the requester and updates local indices. This approach focuses more on object discovery than message reduction. At the cost of an increased message production compared to modified-BFS (because of the update process), the algorithm increases the number of hits. It achieves very high accuracy, enables knowledge sharing and induces no overhead during node arrivals/departures. On the other hand, its message production is very large and only increases with time as knowledge is spread over the nodes. It shows no easy adaptation to object deletions or peer departures. This happens because the algorithm does not utilize negative feedback from searches and forwarding is based on ranking.

Finally, its accuracy depends highly on the assumption that nodes specialize in certain documents [11].

2.2.1.2 Adaptive Probabilistic Search (APS):

In (APS), each node keeps a local index consisting of one entry for each object it has requested per neighbor. The value of this entry reflects the relative probability of this node's neighbor to be chosen as the next hop in a future request for the specific object. Searching is based on the deployment of k independent walkers and probabilistic forwarding. Each intermediate node forwards the query to one of its neighbors with probability given by its local index. Index values are updated using feedback from the walkers. If a walker succeeds (fails), the relative probabilities of the nodes on the walker's path are increased (decreased). The update procedure takes the reverse path back to the requester and can take place either after a walker miss (optimistic update approach), or after a hit (pessimistic update approach).

APS exhibits many plausible characteristics as a result of its learning feature. Every node on the deployed walkers updates its indices according to search results, so peers eventually share, refine and adjust their search knowledge with time. Walkers are directed towards objects or redirected if a miss or an object deletion occurs.

APS is also very bandwidth-efficient, achieving very similar levels with Random Walks. It induces zero overhead over the network at join/leave/update operations and displays a high degree of robustness in topology changes. The s-APS modification adaptively switches between the optimistic and pessimistic approaches and further reduces the amount of messages.

Finally, the algorithm's majority of discovered objects are located close to the requesters. These advantages are mainly seen when many different peers contribute with big workloads. This is because APS gains from knowledge build-up and increased peer cooperation [7].

Figure 2.4 illustrates how the search process works. Peer A issues a query for an object stored on peer F . Two walkers are deployed. Peer A made an optimistic guess. The initial values of all index entries for this object are 30. One walker $w1$ takes the path $A \rightarrow B \rightarrow F$. The other one $w2$ takes the path: $A \rightarrow C \rightarrow D \rightarrow E$. During the search, each node except the last node on the query paths increases the index value(s) for this object and the chosen neighbor(s) by 10. Since the optimistic approach is employed and $w2$ fails, the index values on the query path for $w2$ will be decreased by 20 so that the final index values are smaller than the initial index values. When the subsequent request for the same object is initiated at or forwarded to A , the neighbor B will be chosen with the probability $4/9$ ($40/(20+30+40)$), C with the probability $2/9$, and G with the probability $3/9$.

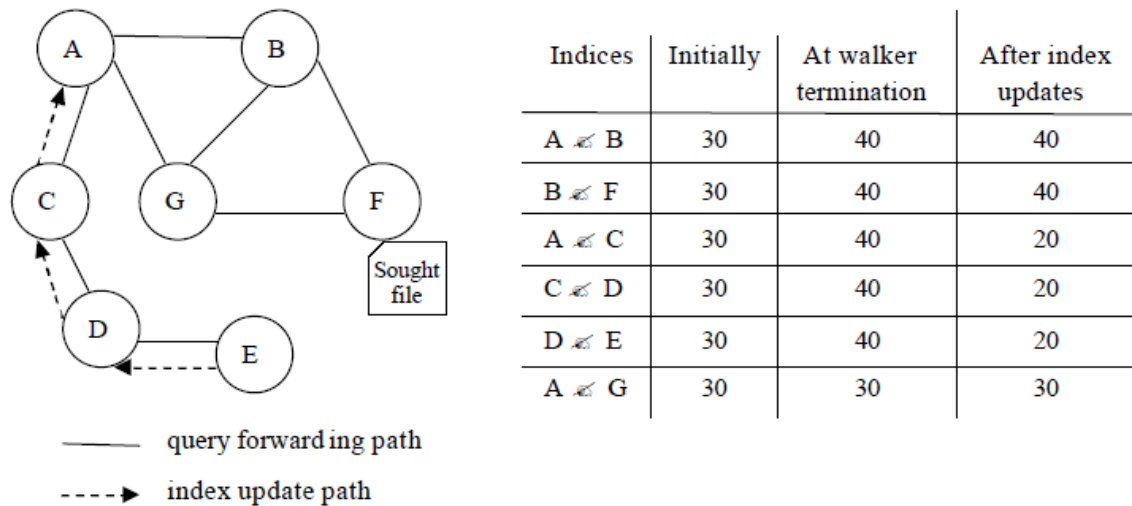


Figure 2.4 Adaptive Probabilistic Search example.

2.2.1.3 Distributed Resource Location Protocol (DRLP):

Nodes with no information about the location of a file forward the query to each of their neighbors with a certain probability. If an object is found, the query takes the reverse path to the requester, storing the document location at those nodes.

In subsequent requests, nodes with indexed location information directly contact the specific node. If that node does not currently obtain the document, it just initiates a new search as described before.

This algorithm initially spends many messages to find the locations of an object. In subsequent requests, it might take only one message to discover it. Obviously, a small message production is achieved only with a large workload that enables the initial cost to be amortized over many searches. In rapidly changing networks, this approach fails and more nodes have to perform blind search. This also affects the number of hits: If many blind searches are made, then many results are found; if many direct queries take place, then only one replica is retrieved. So, this scheme is very dependent on network/application parameters [12].

2.2.1.4 Intelligent search:

The query type considered in the work is the *keyword query*: a search for documents that contain desired keywords listed in a query. A query is represented using a keyword vector. This technique consists of four components:

- 1-search mechanism.
- 2-profile mechanism.
- 3-peer ranking mechanism.
- 4-query similarity function.

When the querying node initiates a query, it does not broadcast the query to all its neighbors. Instead, it evaluates the past performance of all its neighbors and propagates the query only to a subset of its neighbors that have answered similar queries before and therefore will most likely answer the current query. On receiving a query message, a neighbor looks at its local data store. If the neighbor has the desired documents, it returns them to the querying node and terminates. Otherwise, the neighbor forwards the query to a subset of its own neighbors that have answered similar queries before. The query forwarding stops when the maximum TTL is reached [11].

3. Conclusion:

This chapter discusses various searching techniques in peer-to-peer networks (P2Ps). First, the concept of peer-to-peer networks is introduced and different schemes are classified. Next, searching strategies in unstructured P2P systems, blind search in P2P systems, and informed search P2P in systems in the third chapter we present our proposal and the results of comparison. .

CHAPTER 3

DESIGN AND IMPLEMENTATION

1. Introduction:

In this chapter, we will describe the design and experimental setups of our proposal and compare it to the original scheme Iterative Deepening Search (IDS) that we succeed to implement it.

2. Proposed solution

2.1. General explanation:

To improve the performance and development of the method Iterative Deepening Search that by reducing the disadvantages mentioned in Chapter II, by using the idea of Reinforcement Learning and applied to each element in the network (Node).

The Research be through the desired file name, by creating a table named routing table for each file. And transmitter in the search process to be Neighbors.

we identify neighbors the ability to give a positive result for the desired file from this transmitter's a table.

We named our proposal by **RLIDS** shortcut for **R**einforcement **L**earning **I**terative **D**eepening **S**earch.

2.2. Routing table

At the beginning of search, the table RT of each node is empty. After creating query, a table RT for this query is created. This table is filled by negative values, followed by the process of neighbors of this node. Since the table RT is empty, the neighbor receivers are chosen randomly.

After receiving the result, the table of the query is updated on all its path.

This update consists of two stages:

1. Deleting failed elements that do not respond, and keeping the active ones
2. Appointing or entering the new nodes that responded to the query.

Example:

1. Routing table of **Q1**, filled with equal values "0" for all related neighbors that connected with the querying node.
2. We send the query to all neighbors, the same Process carried out by the neighbors.
3. The process is repeated in each NODE until the Q1 up to the NODE has no Neighbors.

4. Upon arrival the Q1 to PC04 send ACK “file found” to PC and PC01 send ACK to the querying node. When Q1 arrived to PC06 replayed with NACK (file not found) to PC02 and PC02 replay NACK to querying node.

5. The Updates occurs when the reference NACK, be along the path reference NACK, this sense that each NODE receives NACK that updates its table like in case (02) when pc04 lose the data so send a NACK and the nodes that receive the NACK updates its RT of the QUERY.

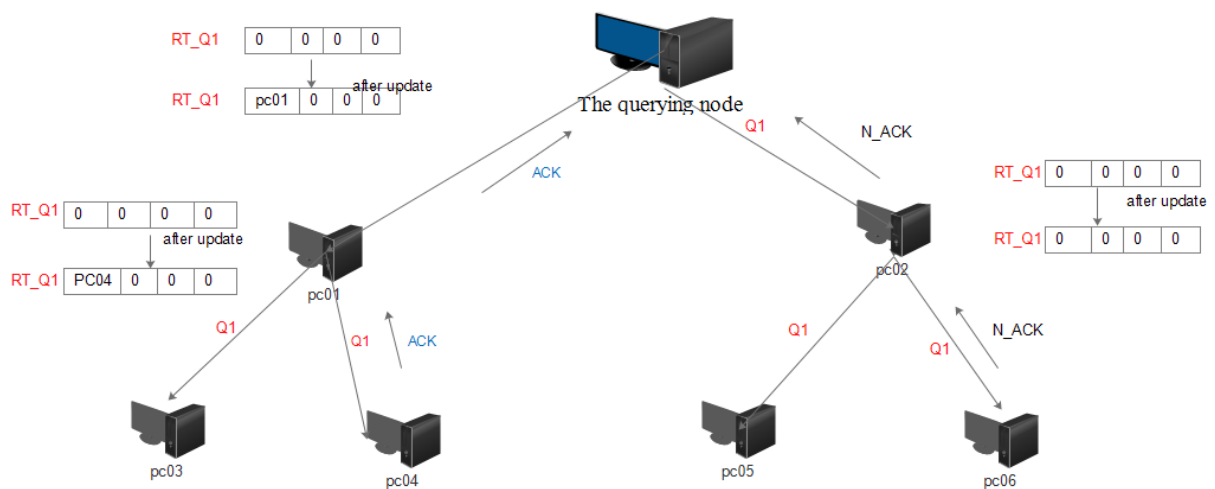


Figure 3.1 RLIDS example case (01).

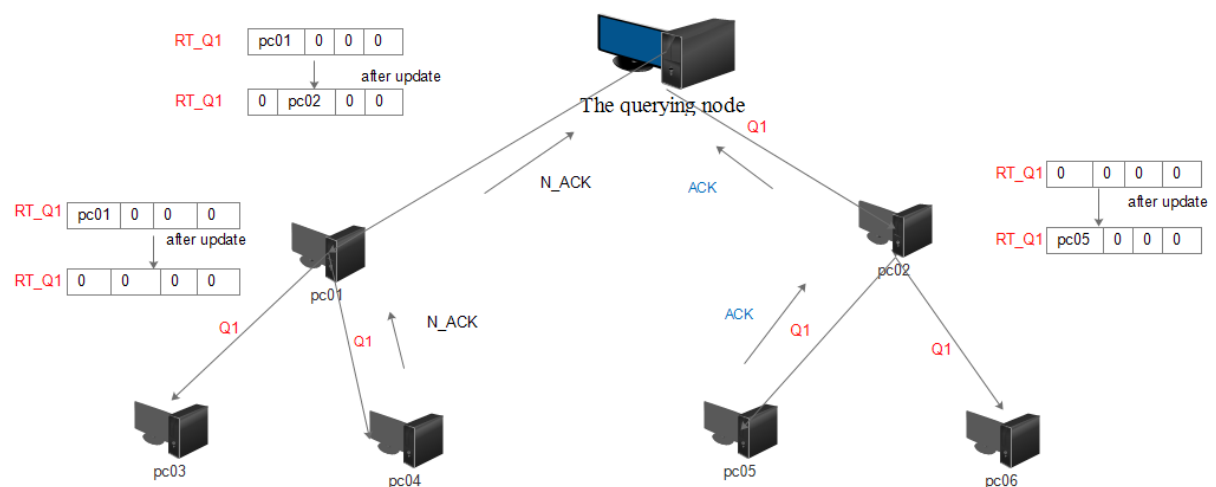


Figure 3.2 RLIDS example case (02).

2.3. Diagrams:

2.3.1. Use case diagram

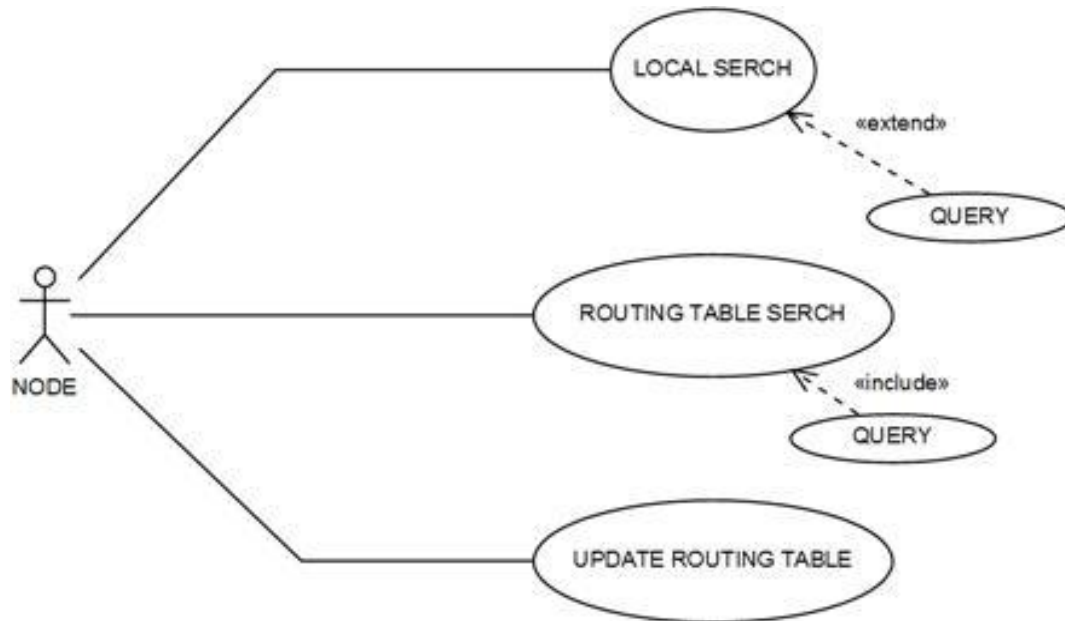


Figure 3.3 Use case diagram.

Scheme (*Fig 3.3*) represents tasks that can do the node:

1. The node do local search and when it finds a neighbor to send the query if not find a neighbor will not send.
2. The node do search in the routing table then send the query to all neighbors
3. The node do update routing-table in the last **Nack or Ack**

2.3.2. Sequence diagram

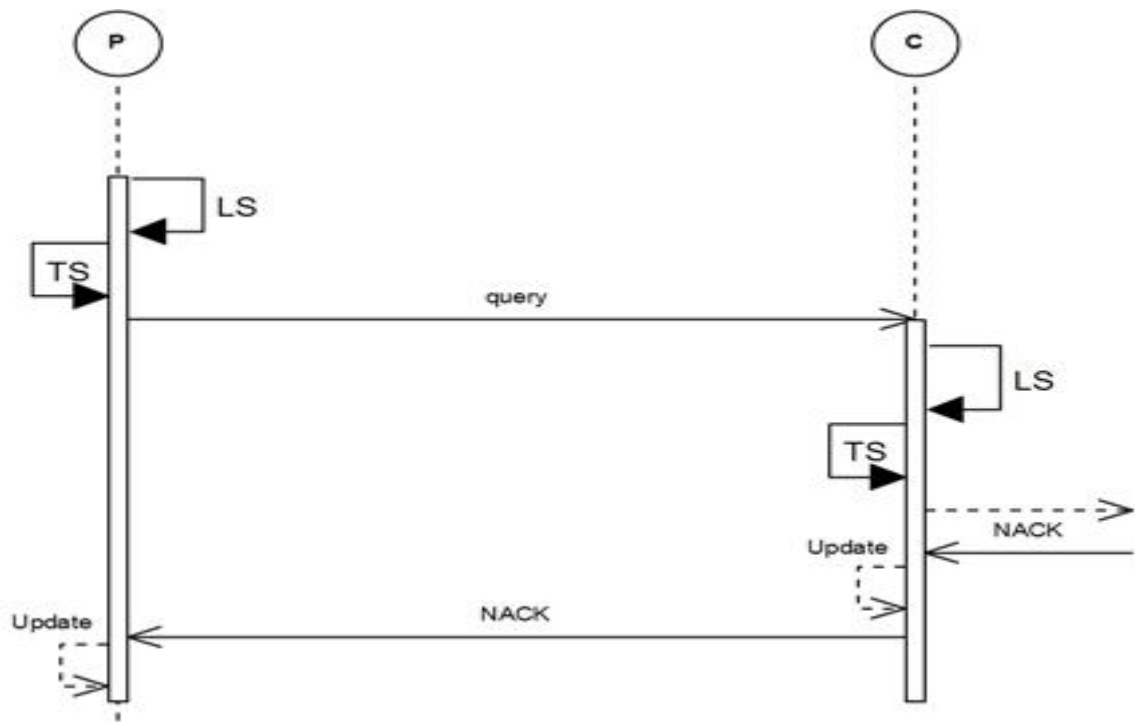


Figure 3.4 Sequence diagram search technique.

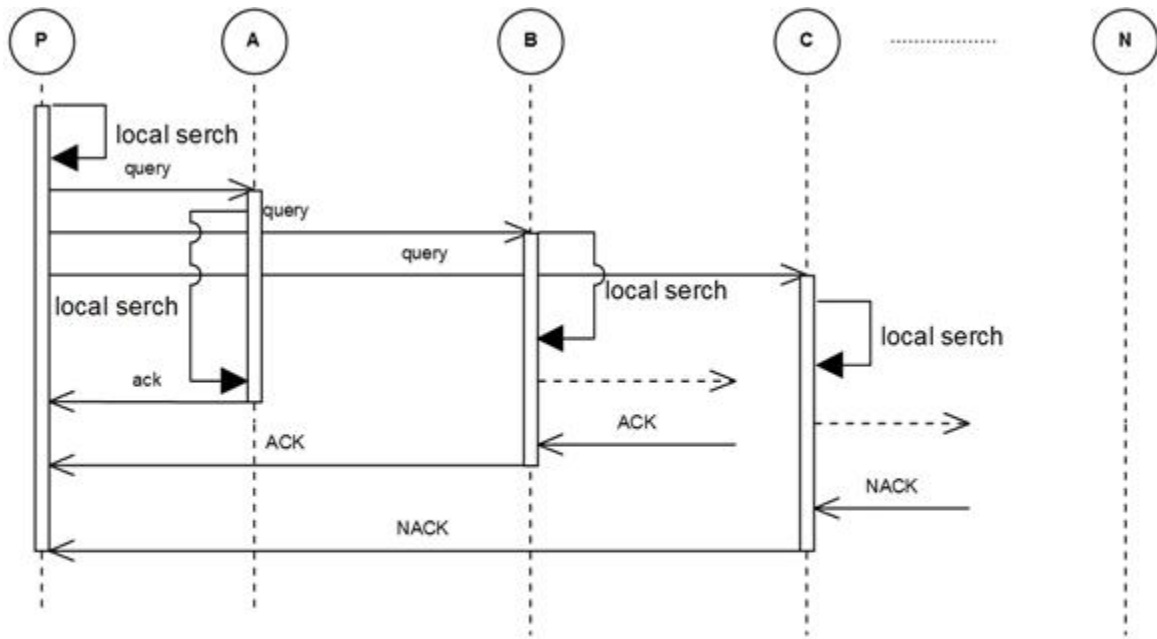


Figure 3.5 Sequence diagram of Messages exchanged.

Fig3.4, Fig 3.5 represents sequence diagram of different search technique and how the update process works. Also various messages are sent. As we see, there are two processes: LS (local search) means search in local files for this node, and TS (table search) means use the routing table to select the neighbor to whom the query will be sent. **ACK** means a successful search, and **Nack** means a failed search. Finally, the update process works when receiving NACK.

2.4. Algorithm:

RLIDS is composed of two core methods, Procedure Route_Query and update table,

- RT.add: add new routing table
- RT.Calcul_W: calculate the number of Query passed by routing table
- RT.Set_N: add new neighbor N in RT of Q
- RT.Sup_N: Remove an exist neighbor N of the RT

/////Q.key is an id of query, S is source peer, and RT is the routing table

/////R is random neighbor

Fonction RLIDS (Q.key, s) {

Pour (depth=0; depth<Max_value; depth++) {

Intger found =Procedure Route_Query (Q.key, Depth)

Si (found>0) return1

}

Procedure Route_Query (Q.key, Depth) {

}

Si (depth>0)

{

//local search

Si (Exist (Q.key, S)) {

Return 1}

Sinon {

Si (not Exist.RT (Q.key, S)) {

RT.add (Q.key, S)

}

Sinon

{

// routing table search

tab_H = RT.Calcul_W (Q.key, S)

I=Tab_H.length ()

Tantque (I > 0) {

Si (Not Route_Query (Q.key, Tab_H [I])

{

//remove the neighbor I because not respond

RT.update (Q.key, S, Tab_H [I], 0)}

Sinon return 1

```

I=I-1
}}
K = neighbor.length
Tantque (K > 0) {
  Si (Route_Query (Q.key, R, depth-1))
    //set the neighbor R because he is respond
    {RT.update (Q.key, S, R, 1)
    Return 1}
  Sinon K=K-1}
} return 0
} sinon return 0
}} end

```

2.4.2 Update table:

```

Procedure RT.update (Q.key, S, N, action) {
  // action is a variable statistic for take the role remove or adding
  Si (action<>0) RT.Sup_N (Q.key, S, N)
  Sinon RT.Set_N (Q.key, S, N)
} end_ procedure

```

3. Implementation:

Our work is to improve research in peer to peer networks by profiteering previous searches of each node within the networks in routing tables of nodes participate in the search process we chose the iterative deepening because its disadvantages which we mentioned in chapter tow.

To do this work we chose the simulator PeerSim Among the existing simulation tools, we chose PeerSim [Pee], which is an open source tool written in Java which has the advantage of already specialized in the study of P2P systems and which has an open architecture and modular which allows it to adapt and to specialize. In addition, it is supported by an active development team, allowing to have at our disposal many basic modules already implemented, including the construction of a realistic physical network topology.

We have implemented RLIDS in PeerSim, and we have implanted also the iterative deepening to make comparison with.

3.1. Simulator PeerSim:

PeerSim is a simulation environment for peer-to-peer networks that is specific to an application which allows an event or simulation cycles. Our developments are moving towards simulation cycles in order to observe normally the performance of our proposal. here we explain briefly operation.

A simulation is defined by a configuration file. The configuration file defines four types of elements:

❖ The Protocols:

This is the main unit of PeerSim. A protocol is a feature associated with a node. Therefore defined in the configuration file, the protocol stack to build on each node. At each cycle, the simulator called on each node, for each protocol, in the order described in the configuration file, the nextCycle () function.

One can also define a protocol to maintain the attributes associated with the nodes. For example, PeerSim defines IdleProtocol; a protocol which holds the vicinity of a node, and whose nextCycle () function is empty.

❖ The initializers:

These are modules launched in early simulation (1 cycle) to initialize the Protocols. In practice, it is used to set the initial network topology (associate each node its neighbors), to distribute the data set (queries and documents).

❖ The Dynamics:

They allow you to apply a dynamic network behavior. This may be to remove links during simulation, turning on or off of the nodes.

❖ The Observers:

They are used to collect the data necessary to build the simulation result. An Observer defined in the configuration file is called at the end of each cycle and performs an observation on the entire network. Implement a P2P system PeerSim therefore involves three steps:

1. Define the components of the protocol layer that we want to simulate what it will initialize on these components and what you want to observe.
2. Construct the corresponding configuration file,
3. Implement protocols needed - if they are not already made available by PeerSim.

3.2. Implementation of the programme:

Public void update_RT (int disired_file, int node, int ID_neighbor, action)

To update the routing table using the desired file ID to enter the RT , and the ID node for update its RT ,and ID_neighbor to know the neighbors, which will replace it if the action equal 0 Or add it if the action equal 1.

Public void RT_add (int disired_file, int id node)

Add a new element to the RT, used id of file for addressing the new element and id node for update his RT

Public int RT_exist (int disired_file, int idnode)

Make sure the presence similar of this desired file in the RT.

Public int [] get_RT (int id_file, int id_node)

Getting the RT of id node for the id file.

```
//*****
```

```
public int RLIDS(Node peer,int protocolID ,Query in) {
    int linkableID = FastConfig.getLinkable(protocolID);
    linkable = (Linkable) peer.getProtocol(linkableID);
    // loops through until a goal node is found
    for ( depth = 0; depth < Integer.MAX_VALUE; depth++) {
        mesgcount2++;
        int found = BFSRT(peer,protocolID,in , depth);
        if (found >0) {
            return 1;
        }
    }
    return -100;}

public int RLBFS(Node peer,int protocolID,Query in,int depth){
    if(depth>0){
        //////////
        int linkableID = FastConfig.getLinkable(protocolID);
        linkable = (Linkable) peer.getProtocol(linkableID);
        //////////
        int nodeId=(int)peer.getID();
        Random rand = new Random();
        //local search ...
        int id_file_local= RandomDistributionInitializer.id_file(nodeId);
        int nodeId_cur=nodeId;
        //
        in.nodeIdsVisited.add(new NodeIds(nodeId));
        boolean fileFound=false;

        int disired_file=in.id_file.FileName ;
        msgcount++;
    }
```

```

        if(id_file_local==in.id_file.FileName ){
            fileFound=true;
        }
        if(fileFound){
            return nodeId;
        }
        else{
            if(linkable.degree()!=0){
                //Begin Lookup of routing table of the desired file in
                //the node current if not we add
                if(RT_exist(disired_file,nodeid_cur)!=1)
                {
                    RT_add(disired_file,nodeid_cur);
                }
            }
            else{
                //Begin Lookup of routing table
                int rout_tab[];
                rout_tab=get_RT(disired_file,nodeid_cur);
                // route the query for all node by using the routing table of that file
                int x=0;
                for (int i = 0; i <linkable.degree() ; i++) {
                    while(rout_tab[x]== 0 &&
x+1<rout_tab.length){x++;if(x>5)break;}

                    if(rout_tab[x]!= 0){
                        Node neighbor= Network.get(rout_tab[x]);
                        int res =RLBFS(neighbor,protocolID,in,depth-1);
                        if(res>0){return 1;}
                    }
                }
            }
        }
    }
}

```

```

else {
    //update the routing table by replace the lose
    nieghbor

    // update--

    update_RT(disired_file,nodeid_cur,(int)neighbor.getID(),0);
    }
    }
    }

    x++;
} //end loop for

}

//in the case of RT lose send the query to random neighbor
if(fileFound==false){
    for(int i= 0; i< linkable.degree(); i++){
        Node
        neighborToAsk=linkable.getNeighbor(rand.nextInt(linkable.degree()));
        int pos=RLBFS(neighborToAsk,protocolID,in,depth-1);
        if(pos>0 )
        {fileFound=true;
        //update the routing table by adding new neighbor
        //update++
        update_RT(disired_file,nodeid_cur,(int)neighborToAsk.getID(),1);
        return 1;

        }

        }

    }

    } //end if
    // the file is found
    if(fileFound){
        return 1;
    }
}

```

```

}

        // in the case of the node leaf
        if((linkable.degree() <= 0)){
            return -100;
        }
    }
    return -100;
} //end method RLIDS

```

3.3. Simulation results:

3.3.1 Scenario of simulation:

We simulated the algorithms using random graphs that have 1000 and 10000 nodes and average degree of neighbor is 5, contains 20 types of files, distributed randomly and repetitively in the network which results in the last 20 * network size.

We Notice consistent results if we change the size of the network (1000 node or 10000 node). There are two methods to compare:

Test 1: Comparison between RLIDS and IDS with different start node (RANDOM) but the same Query.

Test 2: Comparison between RLIDS and IDS with different start node (RANDOM) and different Query.

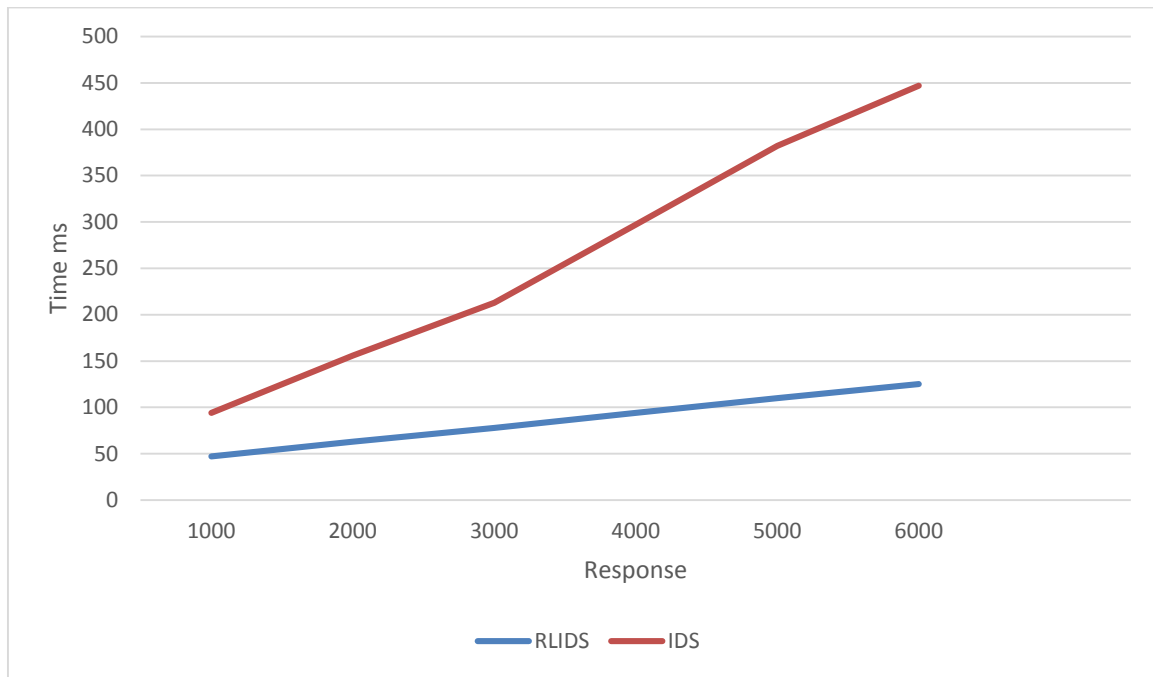


Figure 3.6 The time it takes to get a response (test 1)

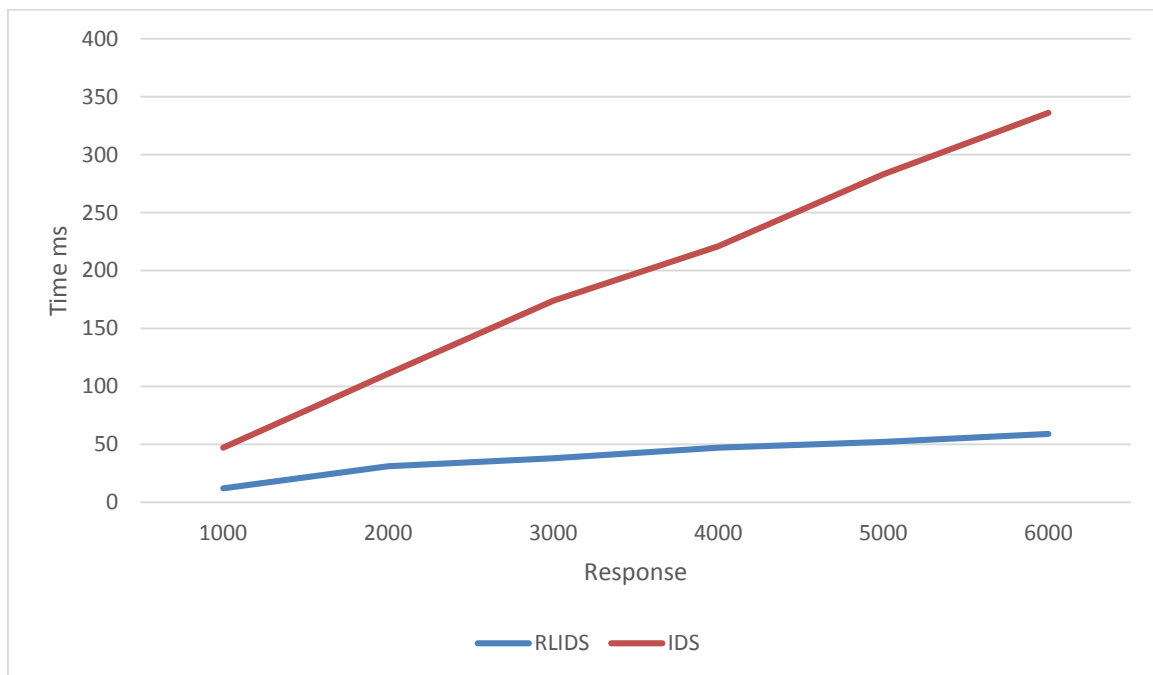


Figure 3.7 The time it takes to get a response (test 2)

Fig 3.6 and **Fig 3.7** represents a time curve for time taken by both processes RLIDS and IDS in the case of different start node(RANDOM) but the same Query or different start node(RANDOM) and different Query Respectively. as we see our proposal achieves a better time to get a response compared to IDS, This is due to random transmitter for a query that is used by IDS Which he abandoned by RLIDS.

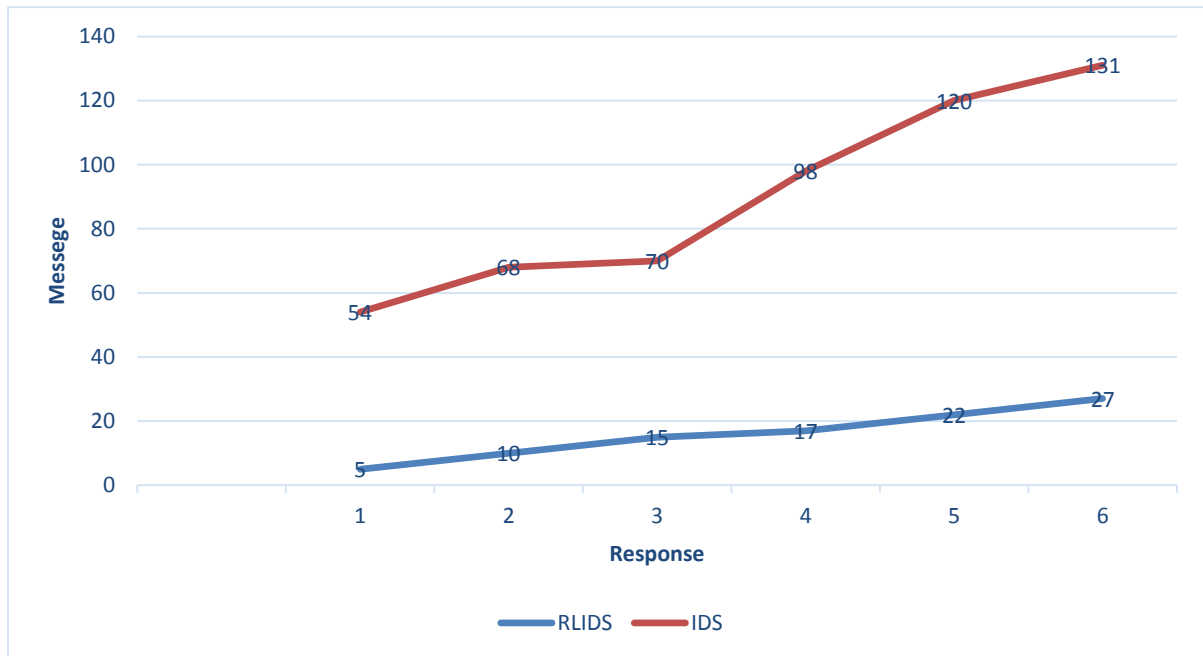


Figure 3.8 Number of messages sent in order to receive a response (test 1)

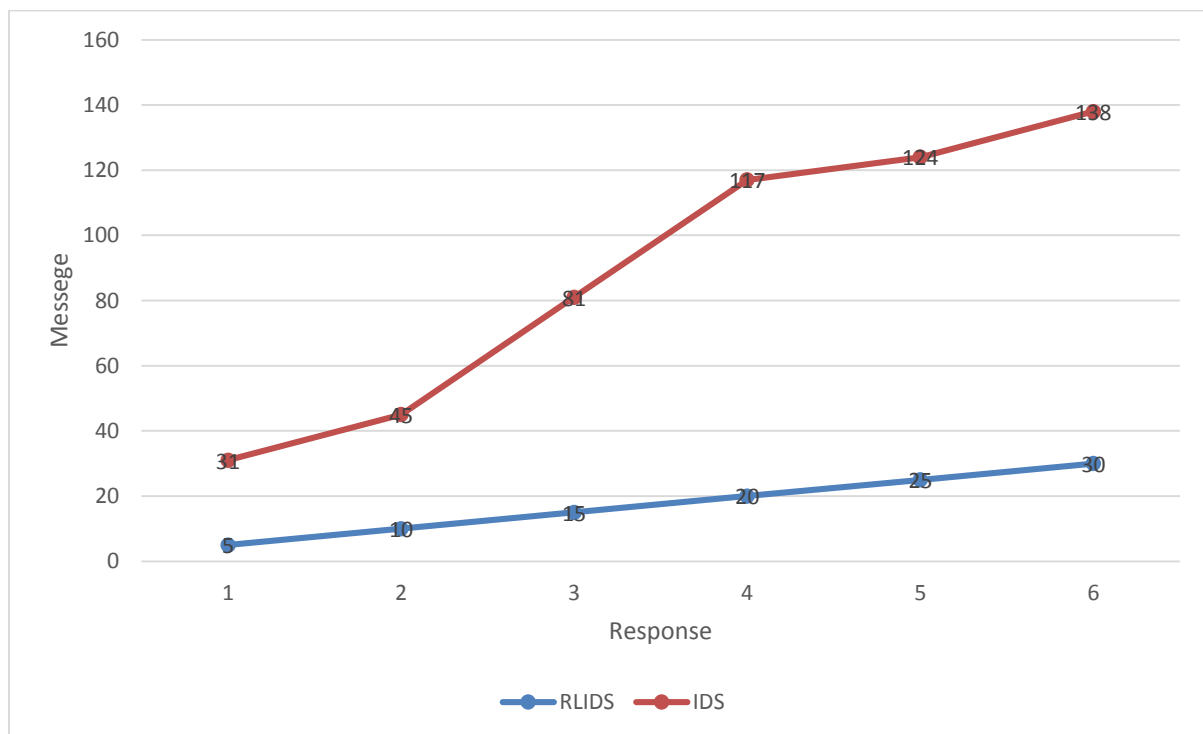


Figure 3.9 Number of messages sent in order to receive a response (test 2)

Fig 3.8 and **Fig 3.9** in the both case of different start node (RANDOM) but the same Query or different start node (RANDOM) and different Query Respectively. We see RLIDS surpass from the start and that's because and after few sending of the messages we see stabilization of the RLIDS graphic curve because its own table has been formed so with that it knows the content of the network which achieve a better and quick search, but KRW continued sending randomly.

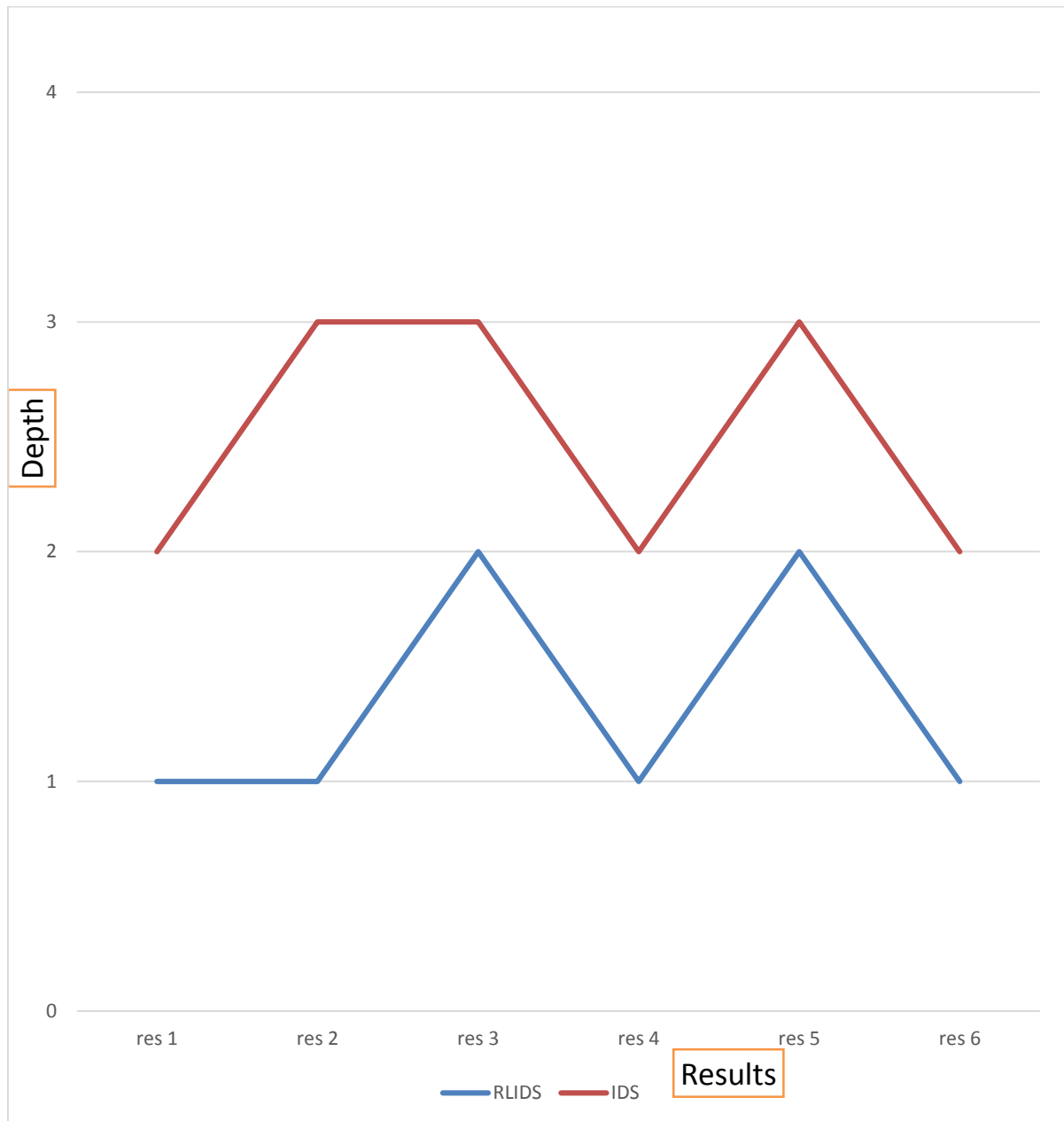


Figure 3.10 the depth necessary to reach the result

Figure 3. 10 Comparison in the depth necessary to reach the results between IDS and RLIDS with deferent start node (RANDOM) but the same Query, we see that RLIDS need less depth than the IDS.

4. Comparison:

In this table we make a comparison between RLIDS and IDS, in number of message send to get a response IDS sends a large number but RLIDS sends far fewer, as well in the time to get response and the depth.

	NBR of MSG	time to get response	depth to get response
RLIDS	+	+	+
IDS	+++	+++	++

Table 3.1 comparison between the RLIDS and IDS

❖ In RLIDS we minimize:

- the time of search
- the messages sent to reach the result
- the depth

5. Conclusion:

After the simulation we found that our proposal perform well then the iterative deepening (IDS) especially in the number of message and time taken , in the beginning of search the tow algorithm start with same manner but the RLIDS benefit and learn from the searches in the network.

General Conclusion:

In this work we have interested by the peer to peer networks, we start by generality about client/server and peer-to-peer networks in the first chapter.

In the second chapter of the different information searching techniques in unstructured peer to peer networks that can be divided on two sets: blind and informed schemes ,we focus on blind schemes in order to ameliorate its performance by the fact of integrate the traffic change and the object consideration in our proposed scheme RLIDS (Reinforcement learning Iterative Deepening Search).

In the third chapter the simulation results show that our proposal perform well then an IDS (Iterative Deepening Search) scheme in NBR of MSG sent the time.we propose, as a future work to compare our proposal with the APS and intelligent search schemes to evaluate more precisely its performance.

Bibliographie:

Books:

[4] Peer-to-Peer Computing principles and application authors: **Vu**, quang hieu, **Lupu**, Mihai, **Ooi**, beng chin.

Article:

- [1] Camar 2009] G. Camarillo, Ed. Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, examples, and Applicability. Request for Comments: 5694, 2004. (Cited on page 5.)
- [2] S. Androutsellis-Theotokis and D. Spinellis. A Survey of Content Distribution Technologies. *ACM Computing Surveys*, 36(4), December 2004. (Cited on page 5.)
- [3] R. Schollmeier. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. *Proceedings of the First International Conference on Peer-to-Peer Computing*, IEEE (2002). (Cited on page 6.)
- [7] D. Tsoumakos, and N. Roussopoulos, "Adaptive probabilistic search in peer-to-peer networks", *Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003.
- [8] B. Yang, and H. Garcia-Molina, "Improving search in peer-to-peer networks", *Proc. of the 22nd IEEE International Conference on Distributed Computing (IEEE ICDCS'02)*, 2002.
- [9] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks", *Proc. of the 16th ACM International Conference on Supercomputing (ACM ICS'02)*, 2002.
- [10] I. Jawhar and J. Wu, "A Two-Level Random Walk Search Protocol for Peer-to-Peer Networks", *Proc. of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics*, 2004.
- [11] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-yazti, "A local search mechanism for peer-to-peer networks", *Proc. of the 11th ACM Conference on Information and Knowledge Management (ACM CIKM'02)*, 2002.
- [12] D. Menascé and L. Kanchanapalli. Probabilistic Scalable P2P Resource Location Services. *SIGMETRICS Perf. Eval. Review*, 2002.
- [13] Wolf, M. (2002). *Speed!: Understanding and Installing Home Networks*. SAMS Publishing 0-672-32186-6.
- [14] P. Ganesan, K. Gummadi, H. Garcia-Molina, Canon in G major: designing DHTs with hierarchical structure, in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS)*, pp. 263–272, 2004

- [15]B. Yang, H. Garcia-Molina, Comparing hybrid peer-to-peer systems, in *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*, 2001
- [18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content addressable network, in *Proceedings of the ACM SIGCOMM Conference*, pp. 161–172, 2001
- [19]D. Karger, F. Kaashoek, I. Stoica, R. Morris, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in *Proceedings of the ACM SIGCOMM Conference*, pp. 149–160, 2001
- [20]A. Rowstron, P. Druschel, and Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems, in *Proceedings of the 18th IFIP/ACM International Conference of Distributed Systems Platforms (Middleware)*, pp. 329–350, 2001
- [21]R. Schollmeier: A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications; Proceedings of the First International Conference on Peer-to-Peer Computing (P2P.01).

Web sites:

- [5] <http://www.napster.com>
- [6] <http://www.gnutella.com>
- [16] <http://boinc.berkeley.edu/trac/wiki/BoincPapers>, 2005
- [17]<http://freenet.sourceforge.net>
- [22] http://www.webopedia.com/TERM/C/client_server_architecture.html (in February 2003).
- [23]Peer-to-Peer; <http://www.peer-topeerwg.org/tech/glossary.html> (in February 2003).

Abstract:

Peer-to-Peer networks have become a major research topic over the last few years. Object location is a major part in the operation of these distributed systems. In this work, we present an overview of several search methods for unstructured peer-to-peer networks, in unstructured P2Ps can be categorized as either blind or informed schemes in a blind search, nodes do not keep information about data location. In an informed search, nodes store some metadata that facilitates the search, so we propose to improve searching in the blind schemes by using the idea of Reinforcement Learning and applied to each element in the network, each node in our proposal RLKRW (Reinforcement Learning Iterative Deepening Search) learn from the past researches other nodes we achieved good results.

Keywords: Peer to Peer network, reinforcement learning, RLIDS, unstructured P2Ps.

المخلص

أصبحت شبكات الند للند موضوع بحث رئيسي خلال السنوات القليلة الماضية , يمثل موقع الكائن جزءاً رئيسياً في تشغيل هذه الأنظمة الموزعة, في هذا العمل نقدم نظرة عامة على العديد من خوارزميات البحث لشبكات الند للند غير المنظمة والتي يمكن تصنيفها إما خوارزمية عمياء أو مستنيرة, في الخوارزمية العمياء لا تحتفظ العقدات بمعلومات حول موقع البيانات اما في الخوارزمية المستنيرة تقوم العقد بتخزين بعض المعلومات الوصفية التي تسهل عملية البحث , لذا نقترح تحسينات في الانظمة المكفوفة باستخدام فكرة تعزيز التعلم وتطبيقها على كل عنصر في الشبكة , كل عقدة في مقترحنا تتعلم من الأبحاث السابقة للعقد الأخرى , حققنا نتائج جيدة بالمقارنة مع الخوارزمية السابقة.

RLIDS

الكلمات المفتاحية : RLIDS, شبكات الند للند, تعزيز التعلم.

Résumé:

Les réseaux Peer-to-Peer sont devenus un sujet de recherche majeur ces dernières années. La localisation des objets est un élément majeur dans le fonctionnement de ces systèmes distribués. Dans ce travail, nous présentons un aperçu de plusieurs méthodes de recherche pour les réseaux peer-to-peer non structurés, dans les P2P non structurés peuvent être catégorisés comme des schémas aveugles ou informés dans une recherche aveugle, les nœuds ne conservent pas d'informations sur l'emplacement des données. Dans une recherche informée, les nœuds stockent des métadonnées qui facilitent la recherche, nous proposons donc d'améliorer la recherche dans les schémas à l'aveugle en utilisant l'idée d'Apprentissage par Renforcement et appliquée à chaque élément du réseau, chaque nœud dans notre proposition RLKRW (Reinforcement Learning Iterative Deepening Search) apprendre du passé recherche d'autres nœuds, nous avons obtenu de bons résultats.

Mots clés: Les réseaux Peer-to-Peer, P2P non structurés, RLKRW