ACKNOWLEDGEMENTS

First of all we would like to thank Allah, the merciful, for giving us the health and the strength to finish this memoir.

It is a great pleasure to me to thank the many people who in different ways have supported my graduate studies and contributed to the process of writing this memoir. First, I would like to thank our supervisor and my teacher **Mr. Mohamed Sahraoui** for his insightful direction and support during the period of writing the memoir. Also I would like to express my gratitude to the management and the staff of our university.

I would like to thank my family and friends for their sincere interest in our work and their moral support. Finally, above all we thank Allah for His guidance again.

Contents

List of figures

List of tables

General introduction

Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes.

Peers make a portion of their resources, such as processing power, disk storage or network bandwidth, directly available to other network participants, without the need for central coordination by servers or stable hosts. Peers are both suppliers and consumers of resources, in contrast to the traditional client-server model in which the consumption and supply of resources is divided.

Peer-to-peer networks generally implement some form of virtual overlay network on top of the physical network topology, where the nodes in the overlay form a subset of the nodes in the physical network. Data is still exchanged directly over the underlying TCP/IP network, but at the application layer peers are able to communicate with each other directly, via the logical overlay links.

Unstructured peer-to-peer networks do not impose a particular structure on the overlay network by design, but rather are formed by nodes that randomly form connections to each other. In particular, 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 raison, many of searching schemes have been proposed to improve the information research mechanism. These schemes can be categorized as either deterministic or probabilistic. The quality of query results in deterministic schemes is low. Probabilistic schemes use simple heuristics that lack the theoretical background to support more efficient results. But this latest solutions did not take in account the dynamic traffic change of the networks caused by the transfer of the data between the nodes. In this work, we propose to improve searching by reinforcement learning (RL), by the fact of integrating in the consideration the dynamic traffic change to be able to learn the best sequence of actions in order to achieve a certain goal.

Our work 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.

CHAPTER 1

PEER-TO-PEER NETWORKS

1. Introduction

The purpose of this chapter is to give an overview for the peer to peer networks; its characteristics and different types in order to present our environment on which we will work in this final stage.

2. Description of peer to peer system:

The Peer-to-Peer (P2P) systems started receiving attention with the introduction of Napster (the first popular file sharing system) in 1999. Since then P2P systems have grown in usage and have become one of the most important applications in the Internet, by many measures. Today, lots of users share various types of files, such as MP3 songs, entire albums, movies, documents, images, software and games.

In P2P systems each peer has a collection of files or data to share. Two peers that maintain an open connection between them are called neighbors as it is represented in Fig1.1.

The number of neighbors of a node defines its out degree or degree. Any peer can pose a query, in order to retrieve interesting content, this peer is called querying peer. When a peer receives a query, first the query is evaluated against its local data collection and thereafter, if necessary other peers are contacted through its neighbors. Query messages are forwarded only

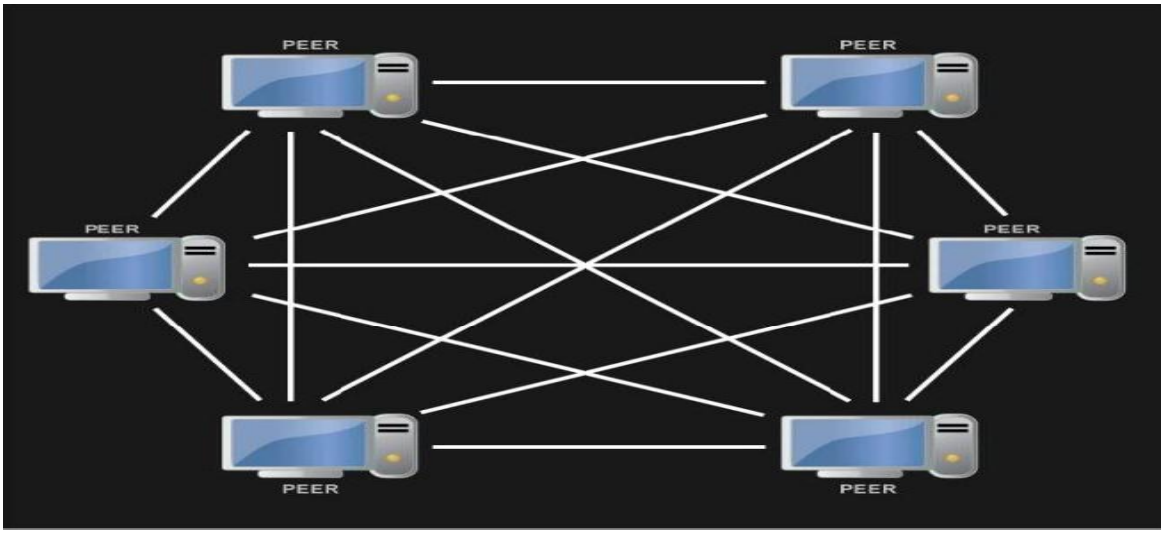


Figure 1. 1Peer to peer network.

Between open connections, i.e., neighboring peers. If a message has to be exchanged between two non-neighboring peers, more than one message is required. The cost of this message is considered that it is proportional to the length of the path, also called number of hops. Usually

query results are forwarded back to the querying peer through the reverse path of the query message. Alternatively, some approaches allow the establishment of a temporary connection, in order to transfer the query results directly to the querying peer [15].

3. Characteristics of the Peer-To-Peer Networks:

P2P systems refer to distributed computer architectures that are designed for sharing resources, by direct exchange, rather than requiring a central coordinating server. In P2P systems, the interconnected computers, called peers, are organized in a network in a distributed and self-organizing way and share resources, while respecting the autonomy of peers. The main characteristic of a P2P system is the ability to adapt to peer failures (fault-tolerance) and accommodate a large number of participating peers (scalability), while keeping the performance of the network at an acceptable level and maintaining the peer connectivity [15].

Generally, all peers are equivalent in terms of tasks and functionality they perform, and act both as clients and servers at the same time. Peers participate in the system in a collaborative manner, by performing tasks such as connecting to other peers, routing

messages to other peers and searching for content. With respect to the peer autonomy, each peer maintains and controls its own content or resources.

Even though different kinds of resources may be shared over a P2P network, such as CPU or storage, P2P systems emerge as a powerful model for organizing and searching of huge amounts of data distributed over independent sources. Applications such as file-sharing, distributed database and digital libraries gain from such an architecture.

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

Peer-to-Peer Networks	**Client/Server Networks**
Each PC is an equal participant on the network	One PC acts as the network controller
PCs are not reliant on one PC for resources such as the printer	One PC controls access to network resources
Access to the network is not centrally controlled	Network access and security are centrally controlled
Can operate on a basic PC operating system	Need a special operating system
Are generally simpler and lower cost	Are generally more complex but give the user more control

Table 1. 1 Peer-to-Peer Networks Vs Client/Server Networks

As shown in Table 1.1, there are definite differences between networks set up as peer-to-peer and those set up as client/server. It should be stated that most home networks today are set up as peer-to-peer, because this network type is simpler and works great for the needs of the home user. Because most home networks today are set up to perform basic but important tasks such as sharing an Internet connection or multiplayer gaming. [16]

5. Peer-To-Peer Architectures:

P2P systems can be classified using a many of criteria as the continent based criteria and the centralization degree one. Based on the way the content is located in the network, with respect to the network topology, we can classify the peer to peer systems into three categories: **purely unstructured**, **semi unstructured** and **structured** networks.

5.1. Purely unstructured P2P systems

In purely unstructured P2P, or purely distributed P2P, each peer maintains a limited number of connections (also called links) to other neighboring peers in the network. Searching in an unstructured P2P environment usually leads to either flooding queries in the network using a time-to-live (TTL) or query forwarding based on constructed routing indices that give a direction for the search. Examples of such purely unstructured P2P networks include Gnutella4[19] and Freenet [10].

5.1.1. Gnutella4 System:

Gnutella 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

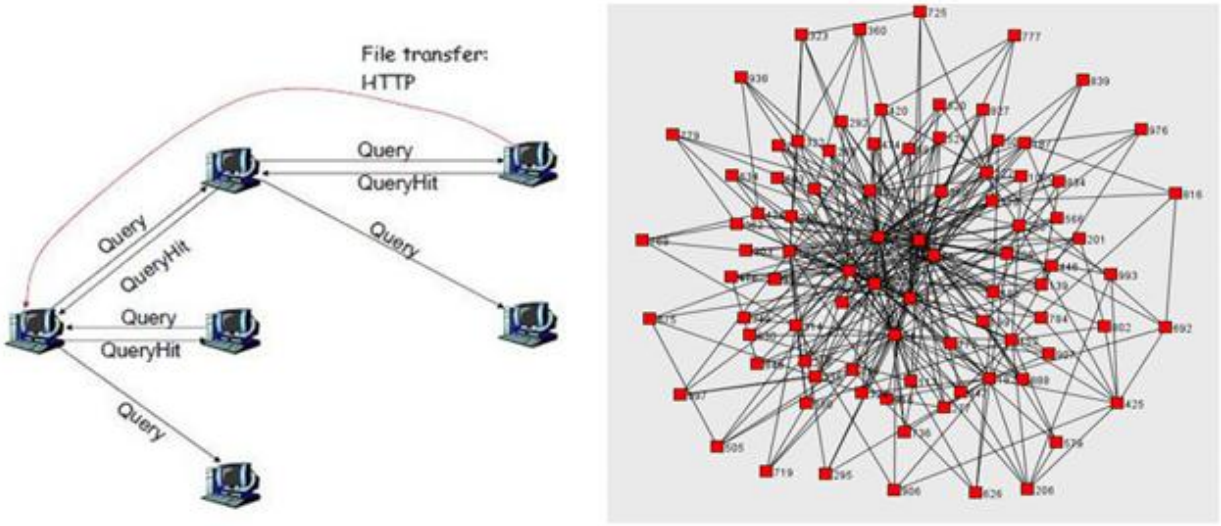


Figure 1. 2Gnutella network.

The packet reached a predetermined number of hops from the sender (maximum 7).

5.2. Semi unstructured P2P systems

In semi unstructured P2P or hybrid decentralized indexing systems, namely super-peer infrastructures, harness the merits of both centralized and purely distributed architectures. Superpeer networks tackle the scaling and ”single-point-of failure” problems of centralized Approaches, while exploiting the advantages of the completely distributed approach, where each peer builds and maintains an index over its own files. These systems are similar to purely decentralized systems, but some of the peers have a more important role, and are responsible to maintain the information available at their associated peers and facilitate the interaction between peers. If only the superpeers are considered, they act as a purely decentralized P2P system. Examples of such of such semi unstructured P2P networks include Gnutella6[18] and Bittorant [12].

5.2.1. Gnutella6 system:

Since version 0.6 (2002), gnutella is a composite network made of leaf nodes and ultra nodes (also called ultrapeers). The leaf nodes are connected to a small number of ultrapeers (typically 3) while each ultrapeer is connected to more than 32 other ultrapeers. With this higher outdegree, the maximum number of hops a query can travel was lowered to 4.

Leaves and ultrapeers use the Query Routing Protocol to exchange a Query Routing Table (QRT. A leaf node sends its QRT to each of the ultrapeers it is connected to, and ultrapeers merge the QRT of all their leaves plus their own QRT (if they share files) and exchange that with their own neighbours. Query routing is then done by hashing the words of the query and seeing whether all of them match in the QRT. Ultrapeers do that check before forwarding a query to a leaf node, and also before forwarding the query to a peer ultra node provided this is the last hop the query can travel. If a search request turns up a result, the node that has the result contacts the searcher.
In the classic gnutella protocol, response messages were sent back along the route the query came through, as the query itself did not contain identifying information of the node.

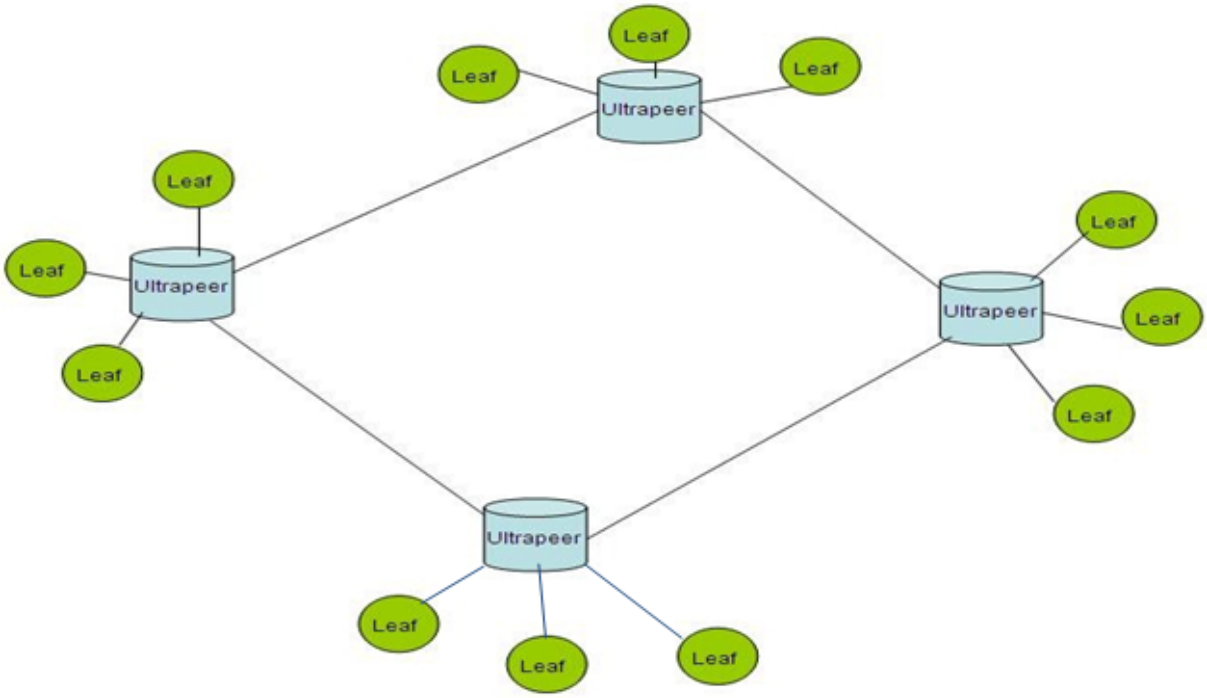


Figure 1. 3 Gnutella 6

5.3. Structured P2P systems:

In structured P2P systems, a hash function is used in order to couple keys with objects. Then a distributed hash table (DHT) is used to route key-based queries efficiently to peers that hold the relevant objects. In this way, object access is guaranteed within a bounded number of hops. Examples of popular structured P2P systems are Chord [11], CAN [8], Pastry [9], Tapestry [3].

5.3.1. Chord system:

Chord is a protocol for a peer-to-peer distributed hash table. A distributed hash table stores key-value pairs by assigning keys to different computers (known as "nodes"); a node will store the values for all the keys for which it discover the value for a given key by first locating the node responsible for that key [11].

Using the Chord lookup protocol, nodes and keys are arranged in an identifier circle that has at most 2^m nodes, ranging from 0 to 2^{m-1}. (m should be large enough to avoid collision.)

Each node has a successor and a predecessor. The successor to a node is the next node in the identifier circle in a clockwise direction. The predecessor is counter-clockwise. In order to complete the circle, Chord take as predecessor of node 0 a node 2^{m-1} .

Depending on the application using Chord, that node might be responsible for storing a value associated with the key. Chord uses a variant of consistent hashing [11] to assign keys to Chord nodes. Consistent hashing tends to balance load, since each node receives roughly the same number of keys, and involves relatively little movement of keys when nodes join and leave the system.

To avoid the linear search above, Chord implements a faster search method by requiring each node to keep a *finger table* containing up to m entries. The i^{th} entry of node n will contain successor $((n+2^{i-1}) \bmod 2^m)$. Every time a node wants to look up a key k, it will pass the query to the closest successor or predecessor (depending on the finger table) of k in its finger table (the "largest" one on the circle whose ID is smaller than k), until a node finds out the key is stored in its immediate successor.

With such a finger table, the number of nodes that must be contacted to find a successor in an *N*-node network is O(log N).

As an example, consider the Chord ring in Figure 1.4. Suppose node 3 wants to find the successor of identifier 1. Since 1 belong to the circular interval [7, 3), it belongs to 3, *Figure 1.4. interval*; node 3 _ therefore checks the third entry in its finger table, which is 0 Because 0 precedes 1 , node 3 will ask node 0 to find the successor Of 1. In turn, node 0 will infer from its finger table that 1’s successor is the node 1itself, and return node 1 to node 3.

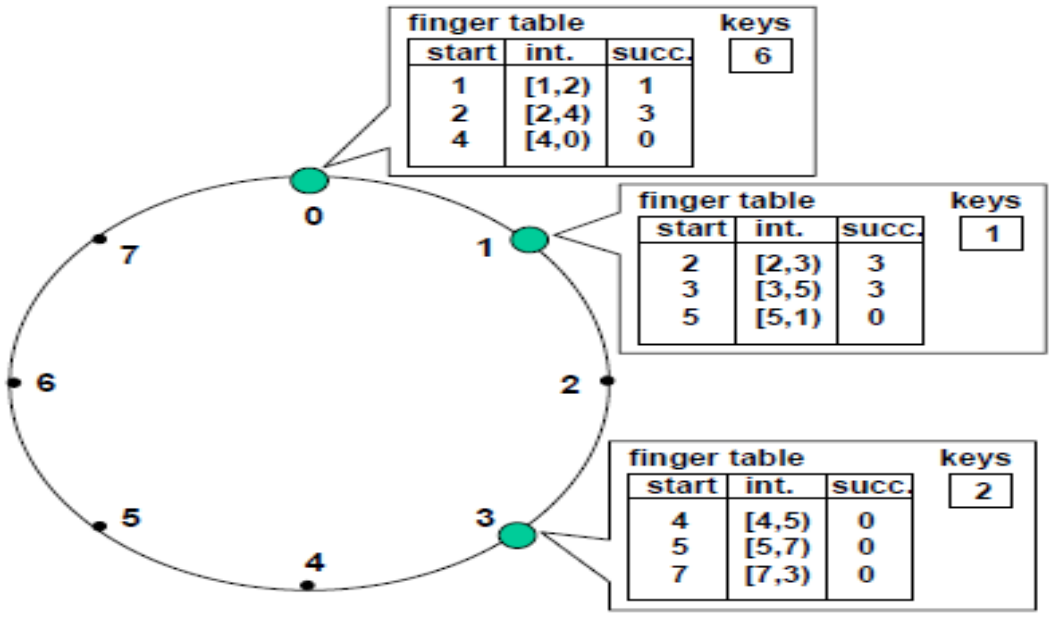


Figure 1. 4 Chord network

6. Advantages and disadvantages of P2P systems:

6.1. Advantages:

1. It is easy to install and so is the configuration of computers on this network,

2. All the ressources and contents are shared by all the peers, unlike server-client architecture where Server shares all the contents and resources.

3. P2P is more reliable as central dependency is eliminated. Failure of one peer doesn't affect the functioning of other peers. In case of Client –Server network, if server goes down whole network gets affected.

4. There is no need for full-time System Administrator. Every user is the administrator of his machine. User can control their shared resources.

5. The over-all cost of building and maintaining this type of network is comparatively very less.

6.2. Disadvantages:

1. In this network, the whole system is decentralized thus it is difficult to administer. That is one person cannot determine the whole accessibility setting of whole network.
2. Security in this system is very less viruses, spywares, Trojans; etc malwares can easily transmit over this P-2-P architecture.
3. Data recovery or backup is very difficult. Each computer should have its own back-up system.
4. Lot of movies, music and other copyrighted files are transferred using this type of file transfer. P2P is the technology used in torrents.

7. Conclusion:

After the exploration of the peer to peer networks making an overview of its important sides, we will focus on the second chapter on one special type named unstructured peer to peer networks in order to study the different solution for its main disadvantage which is the flooding.

CHAPTER 2

INFORMATION SEARCHING TECHNIQUES IN UNSTRUCTURED PEER TO PEER NETWORKS

1. Introduction

In this chapter, we will study the different techniques and schemes proposed in the leterature for unstructured peer to peer networks in order to improve the information searching techniques bay the fact of reducing the number of requests in side and performing the quality of results in term of number and time in the other side. We have choose the unstrustured type for P2P networks because of its speed ability of scaling, simplicity of implimenting and sufring of the flooding.

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

Unstructured P2P systems are extremely resilient to node join-leave, because no special network structure needs to be maintained. Searching in unstructured networks is often based on flooding or its variation because there is no control over data storage.

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 [11] 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.

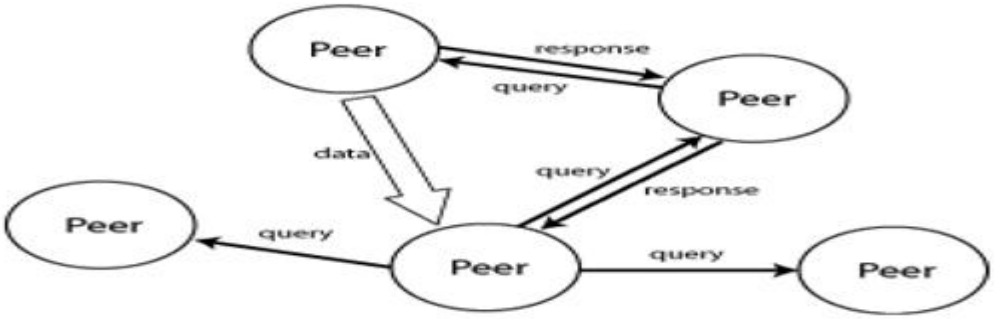


Figure 2. 1 Decentralized and Unstructured Peer to Peer system.

Many alternative schemes have been proposed to address the problems of the original flooding. These works can be classified as deterministic or probabilistic.

2.1. Deterministic schemes:

Deterministic approach, prior information on query path is used for routing and the query forwarding is deterministic. Previous deterministic approaches require a complete walk on the network, since such networks usually lack a central authority. Such approaches hence do not scale well to large networks.

2.1.1. Gnutella:

gnutella[18] the original algorithm uses the spread of breadth-first search (BFS: Breadth-first search) for the discovery of objects, all accessible nodes are contacted within the lifetime value the request (TTL: Time-to-live). this approach not to scale since it produces a high load when it is a question of a significant number of peers.

2.1.2. Iterative Deepening:

For each query Q in the representative set Qrep, our client submitted Q to the live Gnutella network D times (spread over time), where D = 7 is the maximum TTL allowed in Gnutella. Each time we incremented the TTL by 1, so that we submitted Q once for each TTL between 1 and D. For each Query message submitted, we logged every Response message arriving within 2 minutes of submission of the query. For each Response, we log:

- The number hops that the Response message took.
- The response time (i.e., the time elapsed from when the Query message was first submitted, to when the Response message was received).
- The IP address from which the Response message came.
- The individual results contained in the message.

As queries are submitted, our client sent out Ping messages to all its neighbors. Ping messages are propagated through the network in a breadth-first traversal, as Query messages are. When a node receives a Ping message, it replies with a Pong message containing its IP. We sent a Ping

message immediately before every second query, and logged the following information for all Pong messages received in the next 4 minutes:

1. The number of hops that the Pong message took.
2. The IP address from which the Pong came.

In [12], Yang and Garcia-Molina borrowed 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 $D_1 < D_2 < \ldots < D_i$. 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.

For example (Fig 2. 2), 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.

Iterative deepening is tailored to applications where the initial number of data items returned by a query is important. However, it does not intend to reduce duplicate messages and the query processing is slow.

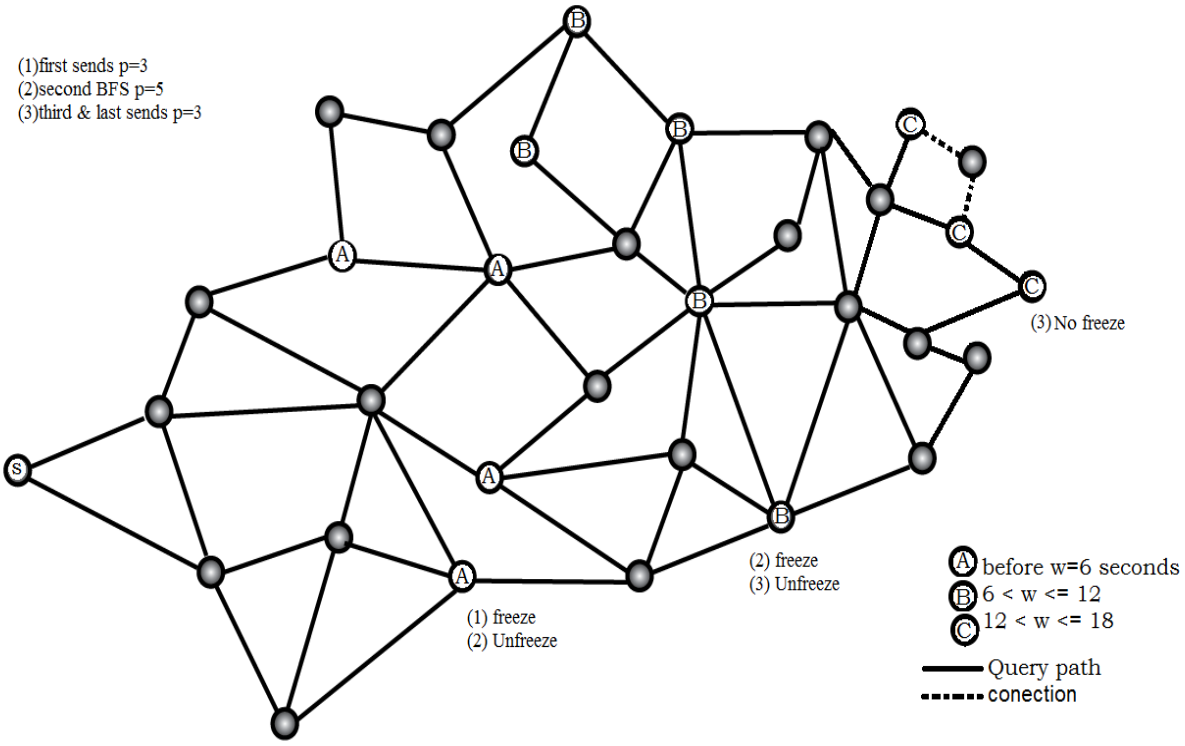


Figure 2. 2 an example of an Iterative deepening.

2.1.3. Attenuated bloom filter based search:

The attenuated bloom filter based search [5] assumes that each stored document has many replic as spread over the P2P network, documents are queried by names. It intends to quickly find replicas close to the query source with high probability. This is achieved by approximately summarizing the documents that likely exist in nearby nodes. However, the approach alone fails to find replicas far away from the query source.

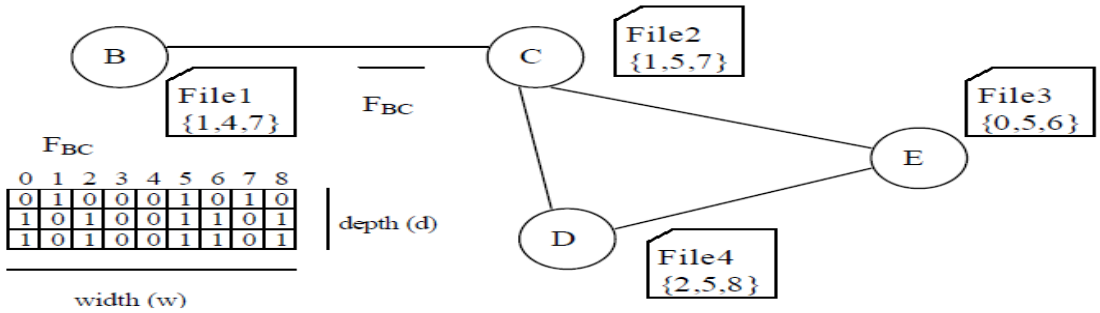


Figure 2. 3 an example of an attenuated bloom filter.

Attenuated Bloom Filters are extensions to bloom filters. An attenuated bloom filter of depth d is an array of d regular bloom filters of the same length w. A level is assigned to each regular bloom filter in the array. Level 1 is assigned to the first bloom filter. Level 2 is assigned to the second bloom filter. The higher levels are considered to be attenuated with respect to the lower levels. Each node stores an attenuated bloom filter for each neighbor. The ith bloom filter in an attenuated bloom filter (depth: d; i = d) for a neighbor B at a node A summarizes the set of documents that will probably be found through B on all nodes i-hops away from A. Fig 2.3 illustrates an attenuated bloom filter for neighbor C at node B. "File3" and "File4" are available at 2-hops distance from B through C. They are hashed to {0, 5, 6} and {2, 5, 8} respectively. Therefore, the second bloom filter contains 1 at bits 0, 2, 5, 6, 8.

2.2. Probabilistic schemes:

In probabilistic approach, the query forwarding is probabilistic , random by choosing randomly neighbors, or is based on ranking, It takes several criteria for rating.

2.2.1. Random walk and *k*-walker random walk:

In the *standard random walk [5]* 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 (Fig 2. 3).

In the *k*-walker random walk algorithm [5], *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. Its main inconvenience is its big variation of performance (Fig 2. 4).

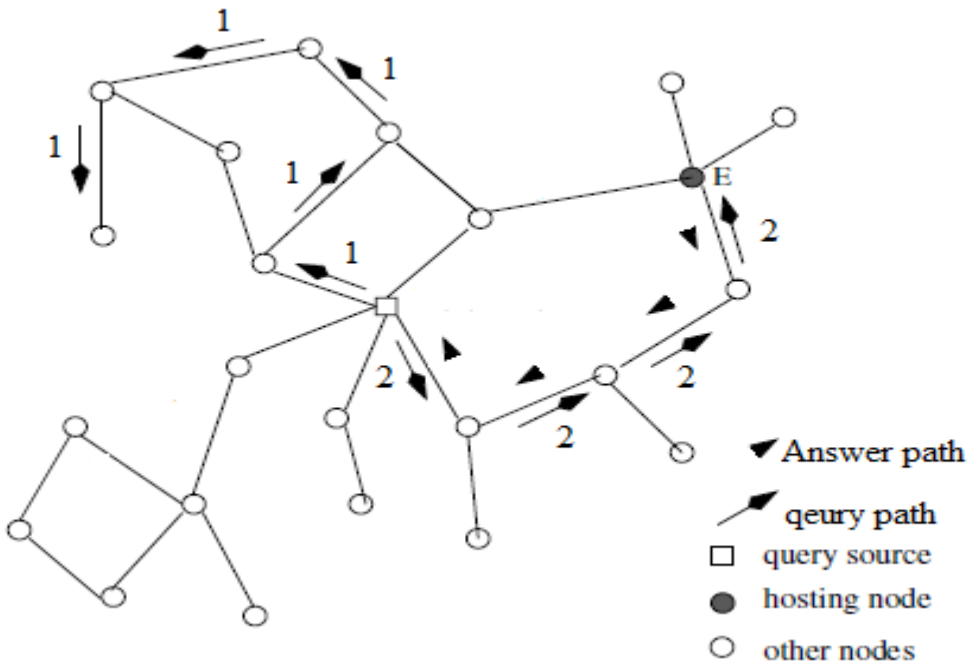


Figure 2. 4 example of two different (1)(2) searches using random walk.

A similar scheme is the two-level random walk [6]. In this scheme, the querying node deploys k1 random walkers with the TTL being l. When the TTL l expires, each walker forges k2 random walkers with the TTL being 2. 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.

Another similar approach, called the modified random BFS, was proposed in. The querying node forwards the query to a randomly selected subset of its neighbors. On receiving a query message, each neighbor forwards the query to a randomly selected subset of its neighbors 7 (excluding the querying node). This procedure continues until the query stop condition is satisfied. No comparison to the k-walker random walk was given in. It is expected that this approach visits more nodes and has a higher query success rate than the k-walker random walk.

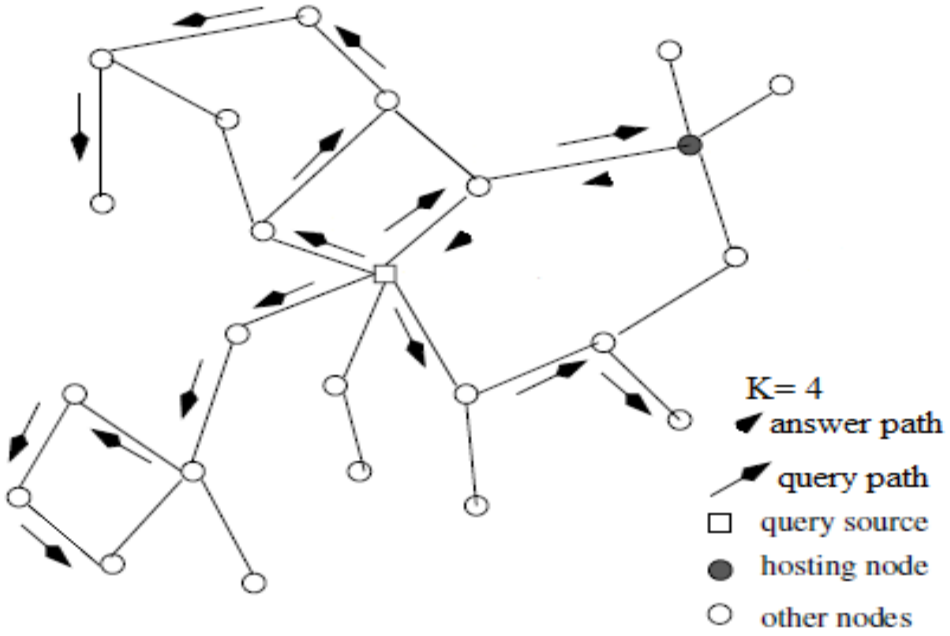


Figure 2. 5 an example of K Random Walk and k = 4.

2.2.4. Routing Indices Based Search:

Routing indices [2] is similar to directed BFS and intelligent search in that all of them use the information about neighbors to guide the search. Directed BFS only applies this information to selecting neighbors of the querying source (i.e. the first hop from the querying source.) The rest of the search process is just as that of BFS. Both intelligent search and routing indices guide the entire search process. They differ in the information kept for neighbors. Intelligent search uses information about past queries that have been answered by neighbors. Routing indices stores information about the topics of documents and the number of documents stored in neighbors.

- Notion:

Routing indices considers content queries, queries based on the file content instead of file name or file identifier. One example of such a content query is: a request for documents that contain the word "networks". A query includes a set of subject topics. Documents may belong to more than one topic category. Document topics are independent. Each node maintains a local index of its own document database based on the keywords contained in these documents.

- The goal:

The goal of a *Routing Index* (RI) is to facilitate a node to select the "best" neighbors to forward queries. A RI is a distributed data structure. Given a content query, the algorithms on this data structure compute the top *m* best neighbors. The goodness of a neighbor is application dependent. In general, a good neighbor is the one through which many documents can be quickly found.

The goodness of a neighbor for a query in compound RI (CRI) is the number of desired documents that may be found through that neighbor. This can be estimated by the following formula:

$$P_{j,q} = ND \times \prod (CRI(ti)/ND) \qquad (2)$$

In the formula(2), $P_{j,q}$ refers to the calculated value of the query q for the node j, *ti* refers to the subject topic that appears in both the query and the CRI table; *CRI(ti)* denotes the value in the intersection of the row for a path and the column for the topic *ti*; *ND* represents the value of the sum of the documents *(docs)* that can be offered by the node i.

- RI types:

Three types of RIs, *compound RI, hop-count RI, and exponentially aggregated RI,* are proposed. They differ in RI index entry structures.

Example CRI:

Compound RI (CRI) stores information about the number of documents in each interesting topic that might be found if a query is forwarded to a single-hop neighbor. A sample CRI at a node B is shown in Table 2.1. Each row in the table describes the number of documents along a specific path and the number of documents on each interesting topic along that path. For example, the first row in the table indicates that if *B* forwards the query to *A*, 1000 documents may be found. Among those documents, 100 are DB documents, 200 are network documents, 400 are theory documents, and there are no language documents.

Path	#docs	Documents in topics			
		Database (DB)	Networks (N)	Theory (T)	Languages (L)
A	1000	100	200	400	0
E	300	60	0	200	100
F	800	0	100	160	200

Table 2. 1 An example of a compound RI at node B.

Use the CRI example for node *B* in Table 2. 1. Assume that *B* receives a query for documents on "networks" and "theory". The goodness of each neighbor for the query is:

A: 1000 x (200/1000) x (400/1000) = 80.

E: 300 x (0/300) x (200/300) = 0.

F: 800 x (100/800) x (160/800) = 20.

Therefore, *B* will select *A* to forward the query because its goodness score is the highest.

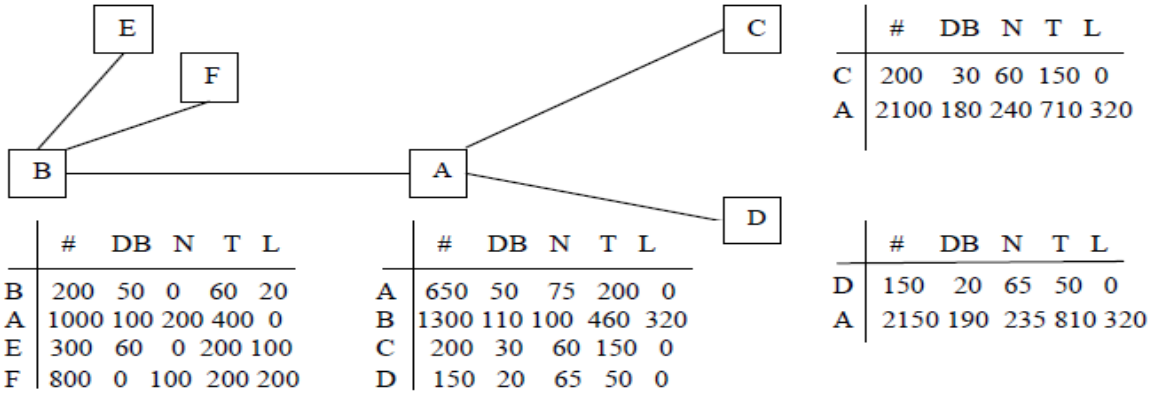


Figure 2. 6 a partial P2P with CRI indices.

(Fig 2.6) shows a partial P2P network and some CRI indices. An additional row is added into the CRI at each node to summarize the local indices in that node. For example, the summary at node *B* indicates that there are 200 documents at *B*; 50 of them are related to database, 60 of them are about theory, and 20 of them are about languages. *B* does not store documents about networks. The CRIs at node *B*, *A*, *C*, and *D* show that node *B* can access 200 network documents via *A*. 75 of them are at *A*, 60 at *C*, and 65 at *D*.

The following shows an example of searching using routing indices. Suppose that the node B initiates a query for the documents about "networks" and "theory". *B* first looks up its local database for the desired documents. If not enough documents are found, it calculates the goodness scores of all its neighbors: A: 80; E: 0; F: 20. A is then chosen as the best neighbor to forward the query. After receiving the query, A first checks its local database and returns all desired documents to B. If the query result is not satisfied, A will then calculate the goodness scores of its neighbors C, D (B is excluded): C : 45, D : 23. A then selects C as the best neighbor to forward the query. C then processes the query and returns all desired data along the query path.

C does not have any other neighbor to forward the query. If the query stop condition is not satisfied, C will return the query back to A. A then forwards the query to its second best neighbor D. This process continues until the desired number of documents is found.

- **Observation:**

The limitation of the hop-count RI is that it does not have information about documents at hop- distance beyond the horizon. The *exponentially aggregated RI (ERI)* solves this problem at the cost of some potential loss in accuracy. The ERI entries store the result of applying the regular- tree cost formula to a corresponding hop-count RI for the topics of interest.

2.2.5. Reinforcement learning:

Reinforcement learning is a learning paradigm concerned with learning to control a system so as to maximize a numerical performance measure that expresses a long-term objective. What distinguishes reinforcement learning from supervised learning is that only partial feedback is given to the learner about the learner's predictions. Further, the predictions may have long term effects through influencing the future state of the controlled system. Thus, time plays a special role. The goal in reinforcement learning is to develop efficient learning algorithms, as well as to understand the algorithms' merits and limitations. Reinforcement learning is of great interest because of the large number of practical applications that it can be used to address, ranging from problems in artificial intelligence to operations research or control engineering, when applying RL to P2P searching, each node is a distributed learner and the P2P network is its environment. in this part we focus on those algorithms of reinforcement learning in p2p system.

2.2.5.1. Directed BFS:

The basic idea of directed BFS approach [12] is that the query node sends the query message to a subset of its neighbors that will quickly return many high-quality results. These neighbors then forward the query message to all their neighbors just as in BFS.

To choose “good” neighbors, a node keeps track of simple statistics on its neighbors, for example, the number of query results returned through that neighbor, and the network latency of that neighbor. Based on these statistics, the best neighbors can be intelligently selected using the following heuristics:

- Select the neighbor that has returned the highest number of results for previous queries.

- Select neighbor that returns response messages that have taken the lowest average number of hops. A low hop-count may suggest that this neighbor is close to nodes containing useful data.
- Select the neighbor that has forwarded the largest number of messages (all types) our client. A high message count implies that this neighbor is stable and it can handle a large flow of messages.
- Select the neighbor with the shortest message queue. A long message queue implies that the neighbor's pipe is saturated, or that the neighbor has died.

By directing the query message to just a subset of neighbors, directed BFS can reduce the routing cost in terms of the number of routing messages. By choosing good neighbors, this technique can maintain the quality of query results and lower the query response time. However, in this scheme only the querying node intelligently selects neighbors to forward a query. All other nodes involved in a query processing still broadcast the query to all their neighbors as in BFS. Therefore, the message duplication is not greatly reduced.

2.2.5.2. 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 [14].

Neighbors are ranked to facilitate the selection. The rank of a neighbor P_i of the peer P_j in terms of the query q is determined by the following formula (1):

$$R_{Pj}(P_i , q) = ? \underset{qi \in A}{(K_{sim}(q_i , q))} \qquad (3)$$

In the formula (3), R denotes the set of queries among the K most similar ones that were answered by peer P_i; $?$ is a configurable parameter used to add more weight to more similar queries. The ranking formula aggregates the similarities of K most similar past queries answered by a neighbor.

2.2.5.3. Adaptive Probabilistic Search:

In the Adaptive Probabilistic Search (APS) [4] , it is assumed that the storage of objects and their copies in the network follows a replication distribution. The number of query requests for each object follows a query distribution. The search process does not affect object placement and the P2P overlay topology.

- **Method in general:**

The APS is based on k -walker random walk and probabilistic (not random) forwarding. The querying node simultaneously deploys k walkers. On receiving the query, each node looks up its local repository for the desired object. If the object is found, the walker stops successfully. Otherwise, the walker continues. The node forwards the query to the best neighbor that has the highest probability value. The probability values are computed based on the results of the past queries and are updated based on the result of the current query. The query processing continues until all k walkers terminate either successfully or fail (in which case the TTL limit is reached).

- **Neighbors select :**

To select neighbors probabilistically, each node keeps a local index about its neighbors. There is one index entry for each object which the node has requested or forwarded requests for through each neighbor. The value of an index entry for an object and a neighbor represents the relative probability of that neighbor being selected for forwarding a query for that object. The higher the index entry value the higher the probability.

- **index entry :**

Initially, all index values are assigned the same value. Then, the index values are updated as follows. When the querying node forwards a query, it makes some guess about the success of all the walkers. The guess is made based on the ratio of the successful walkers in the past. If it assumes that all walkers will succeed (optimistic approach), the querying node pro-actively increases the index values associated with the chosen neighbors and the queried object. Otherwise (pessimistic approach), the querying node pro-actively decreases the index values. Using the guess determined by the querying node, every node on the query path updates the index values similarly when forwarding the query.

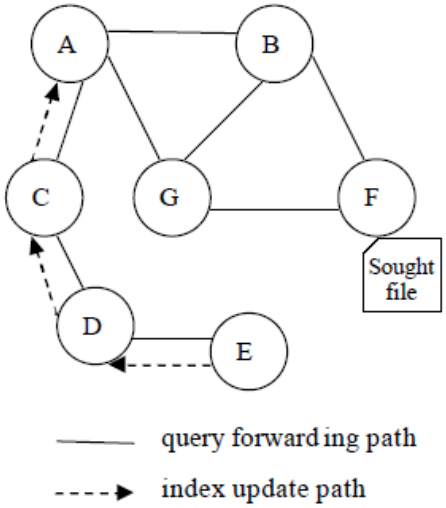
- **index update:**

The index values are also updated when the guess for a walker is wrong. Specifically, if an optimistic guess is made and a walker terminates with a failure, then the index values for the requested object along that walker's path are decreased. The last node on the path sends an update message to the preceding node. On receiving the message, the preceding node decreases the index value for that walker and forwards the update message to the next node on the reverse path. This update procedure continues on the reverse path until the querying node receives an update message and decreases the index value for that walker. If the pessimistic approach is employed and a walker terminates successfully, the index values for the requested object on the walker's path are increased. The update procedure is similar. To remember a walker's path, each node appends its ID in the query message during query forwarding and maintains a soft state for the forwarded query. If a walker A passes by a node which another walker B stopped by before, the walker A terminates unsuccessfully. The duplicate message was discarded.

Example:

(Fig 2.7) 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 -> B -> F. The other one w2 takes the path: A -> C -> D -> 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 w ill be chosen with the probability 4/9 (40/(20+30+40)), C with the probability 2/9, and G with the probability 3/9.



Indices	Initially	At walker termination	After index updates
A ⇙ B	30	40	40
B ⇙ F	30	40	40
A ⇙ C	30	40	20
C ⇙ D	30	40	20
D ⇙ E	30	40	20
A ⇙ G	30	30	30

Figure 2. 7 an example of adaptive probabilistic search.

- **Comparison:**

Compared to the k -walker random walk, the APS approach has the same asymptotic performance in terms of the message overhead. However, by forwarding queries probabilistically to most promising neighbor(s) based on the learned knowledge, the APS approach surpasses the k-walker random walk in the query success rate and the number of discovered objects (table 2.2).

	Performance of message overhead	query success rate	number of discovered objects
The k -walker random walk	+ +	+	+
The Adaptive Probabilistic Search	+ +	+ +	+ +

Table 2. 2 Compared BETWEEN The K -Walker Random Walk AND The APS

APS exhibits many plausible characteristics [7], such as:

- High accuracy
- Low bandwidth consumption
- Large number of discovered objects
- Robust and adaptive behavior in rapidly-changing environments.

2.2.5.4. Improve Searching by Q-learning:

Q-learning:

Q-learning [1] is one simple and widely used RL scheme. It defines a Q function for each state-action pair (s, a). Q(s, a) represents the expected discounted cumulative reward obtained by taking action a at state s. A learner estimates Q(s, a) values iteratively based on experiences as follows. At current state s, take a selected action a, receive an immediate reward r, and observe the new state s . Then the learner updates Q(s, a) according the following formula:

$$Q(s, a) = (1 - \eta)Q(s, a) + \eta(r + \gamma maxbQ(s', b)), \qquad (4)$$

Where η is the learning rate and γ is the discount factor. Both are in the range [0, 1]. $Q(s, a)$ can be implemented as a simple table or a trainable parameterized function.

ISRL [6] **(**Improve Searching by Reinforcement Learning**)** This method applies RL to P2P searching without topology adaptation [13]. They propose two models: basic ISRL (intelligent search by reinforcement learning) and MP-ISRL (multi-path ISRL). The basic version is designed for locating one desired file efficiently. Each node learns the best path to one desired file by exploring new potential paths and exploiting the already discovered paths. To balance exploration and exploitation, each node adapts its exploration coarsely or in a fine-grained manner based on the quality of the learned paths. MP-ISRL aims at finding more than one file efficiently. Each node learns the top-*k* best paths to desired files.

The following are contributed in this method:.

1. Put forward a searching scheme, ISRL, tematically learns the best path to a desired file by reinforcement learning and adapts itself to system dynamics. To the best of our knowledge, this is the first work that applies RL to P2P searching without topology adaptation.

2. Design two models of ISRL, the basic version for locating the best path, and MP-ISRL for finding the top *k* best paths. Important design issues, such as balancing exploration and exploitation are also discussed.

The basic ISRL:

The goal of the basic ISRL is to deliver a query through the best path to a node hosting the desired file. It explores new paths by forwarding query *q* to a random neighbor with probability p_q . It exploits existing paths by sending *q* through the best-so-far path with probability $1- p_q$. The newly explored better path replaces the existing path during path updates. We consider semantic queries for files with similar semantic content.

Example: Fig 2.8 shows an example of the basic ISRL search. The query source is *A*, denoted by an empty square. The desired file is only hosted in node *E*, represented by a solid circle. The arrows indicate query forwarding directions. The number next to each arrowed line refers to the experiment sequence. *A* - *F* are node IDs. The number within curly brackets next to each node ID is the *Q* value for this query. The first trial via *B* is successful, which causes nodes *D*, *C*, *B*, *A* to add new entries for this query with *Q* values as shown in the figure. The second and third trials fail. No update to *Q* values

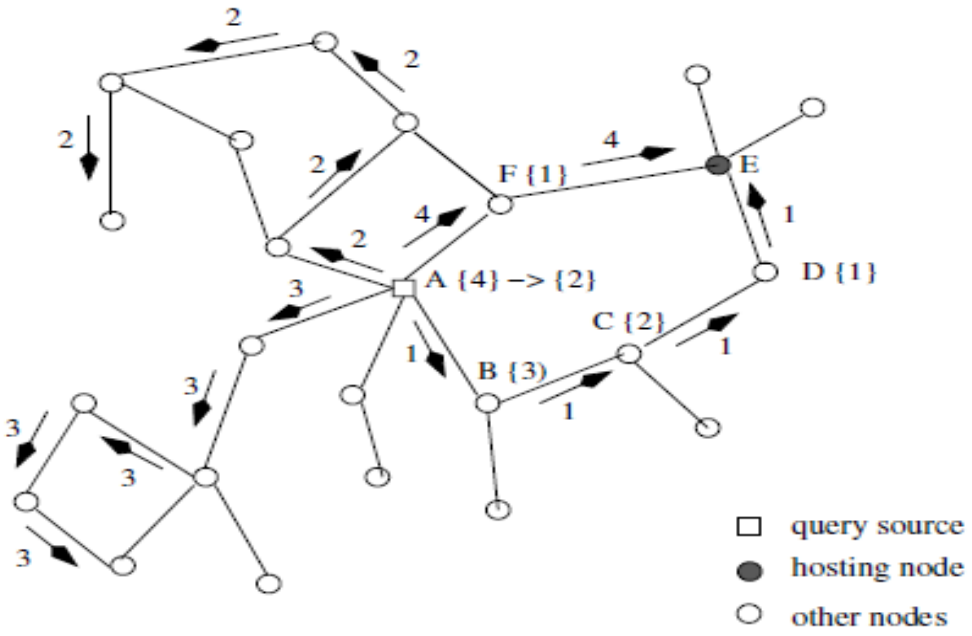


Figure 2. 8 an example of the basic ISRL search.

On the reverse query path, the fourth experiment succeeds and makes node *F* add a new entry with cost 1 for the query. *A* replaces the entry to *B* by the new entry to *F* with lower cost 2. Therefore *A* learns the best path to hosting node *E* through trials.

The basic ISRL corresponds to Q-learning with the following settings: $r = 1$, $\gamma = 1$ and $\eta = 1$. Thus we have

$$Q(s, a) = 1 + min_b Q(s', b)). \qquad (5)$$

Because $Q(s, a)$ represents the path cost, it is to be minimized. r is set to 1 considering that the path cost is in terms of overlay hops. γ is set to 1 because $Q(s, a)$ represents path costs and discounts are unnecessary. η is set to 1 because we can adjust learning through the exploration probability.

3. Conclusion:

After the illustration of the different techniques proposed in order to improve the information searching techniques in unstructured peer to peer networks, we have observed that the probabilistic ones are more beneficial and robust for this type of peer to peer networks specially the schemes based on reinforcement learning. The direct BFS scheme use the highest results history of the neighboring nodes that returns well responses for the new selection without taking in consideration the information identification nor the traffic change caused by the transfer of the data in the term that the other neighboring nodes may returns a more well results then the previous ones. The APS and Intelligent search took in account the object in its statistics but they did not take the traffic change by the fact that they use an incremental value for each object when the neighboring node offer a good response and they decrement this value else which can take a number of detrimental using before that they give the opportunity to other neighboring nodes. Our proposed solution bases on the keeping of the neighboring nodes that returns a well response for each object but when this node cannot offer a good response for one time we change it by another neighboring node that can give as a well response bases on the traffic change in the network. The following chapter, we will explain in details our proposed solution and illustrate the different simulation results.

CHAPTER 3
DESIGN AND IMPLEMENTATION

1. Introduction

In this section, we will describe the design and experimental setups of our proposal and compare it to existing schemes that we succeed to implement it .

2. Proposed solution

2.1. General description

To improve the performance and development of the previous method mentioned in Chapter II, reducing the disadvantages 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, Where ***k*** is equal to one-third of their respective neighbors, for each node.

If N the number of neighbors for each node, then we find k = N / 3, after calculating 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 RLKRW shortcut for Reinforcement Learning k Random Walker

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 calculating K (the number of neighbors to whom the query is sent) which makes a third of the direct 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**,failled to equal values for all with value"0"related to the Neighbors that connected with node P
2. Account K= N/3we find K= 2, we choose 2 Neighbors from Q1 table and sends to A and C, the same Process carried out by A, C .
3. The process is repeated in each NODE until the Q1 up to the NODE has no Neighbors or TTL arrive to 7.
4. Upon arrival to the Q1 to D send ack “file fond” and A replay Ack to P.When Q1 arrived to F replayed with **NACK** (file not fond) and C replay NACK to P
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 .

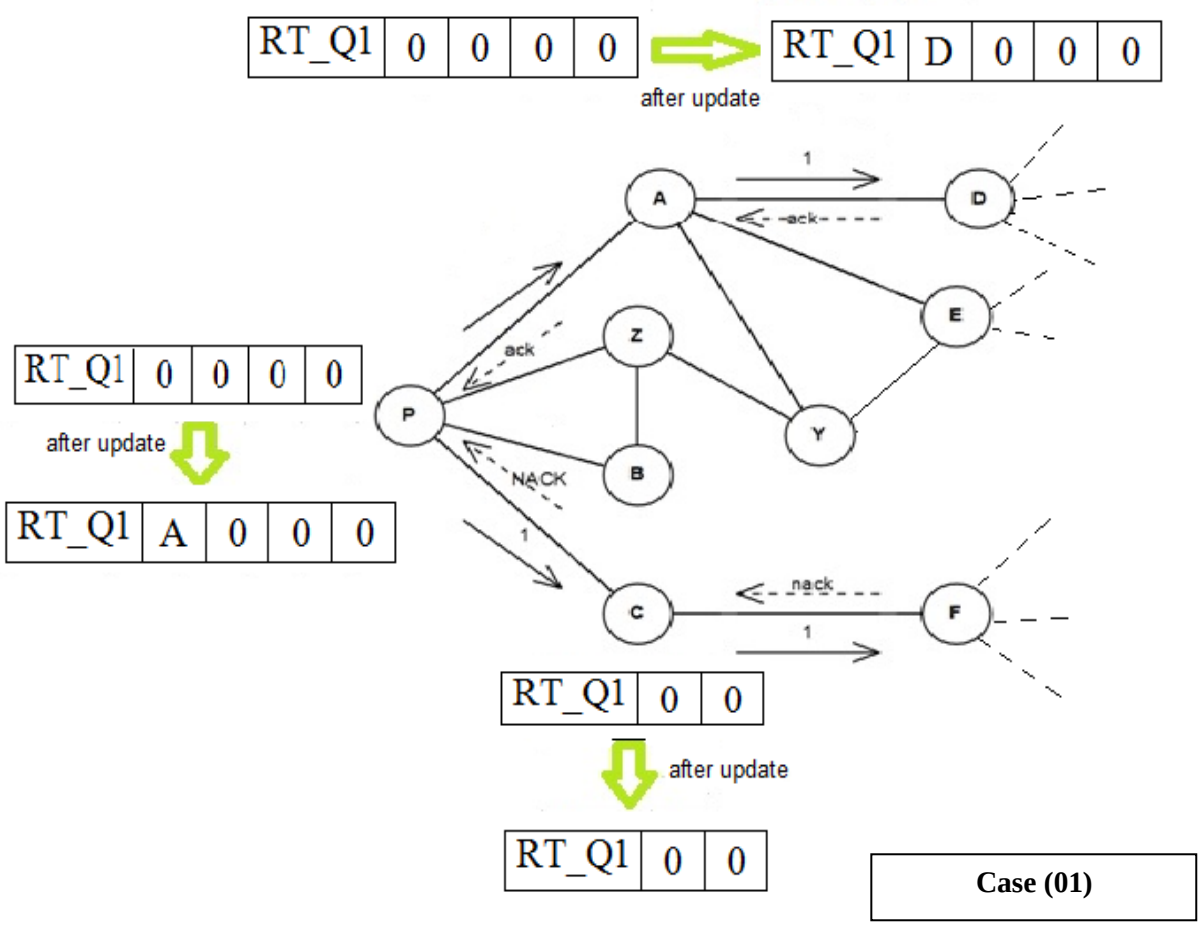


Figure 3. 1 RLKRW example case(01)

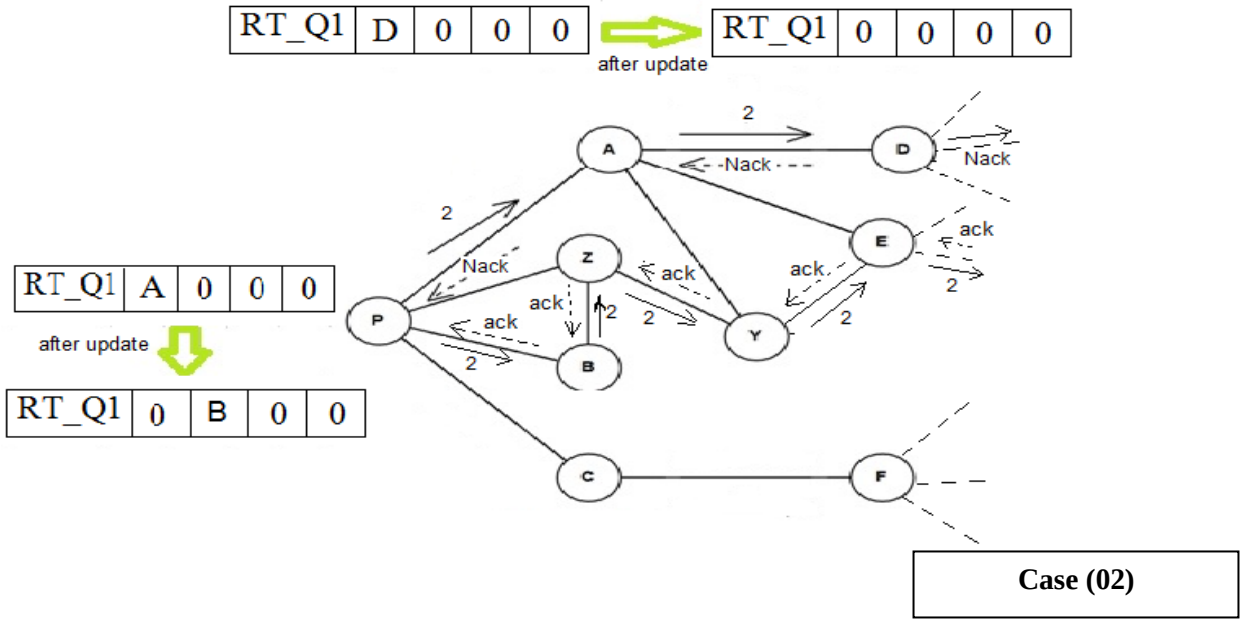


Figure 3. 2 RLKRW example case (02)

2.3. Diagrams:

2.3.1. Use case diagram

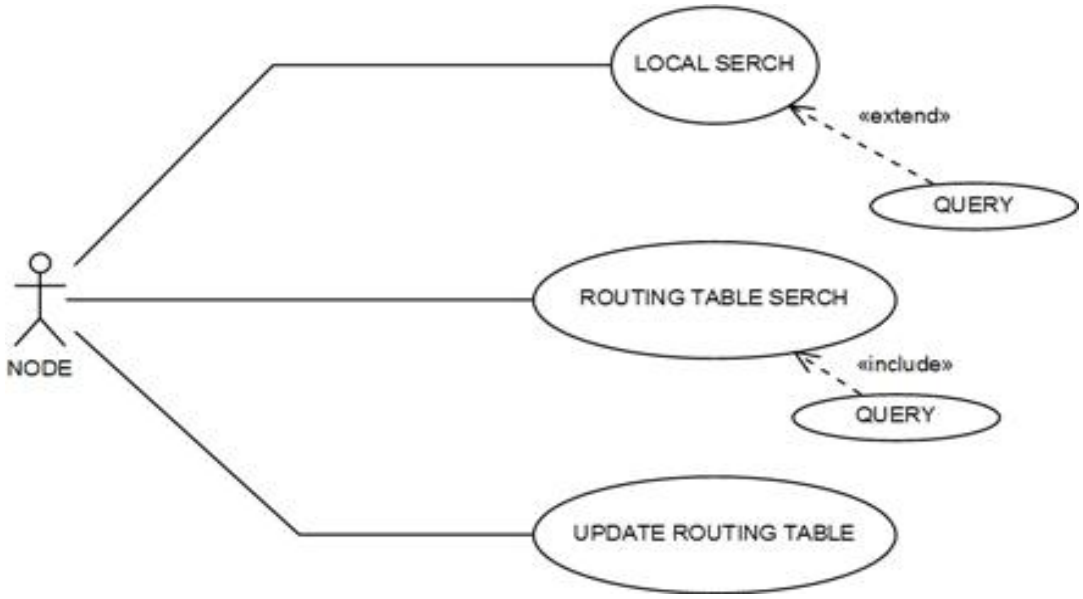


Figure 3. 3 use case diagram

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 k query to neighbors
3. The node do update routing-table in the last **Nack or Ack**

2.3.2. Sequence diagram

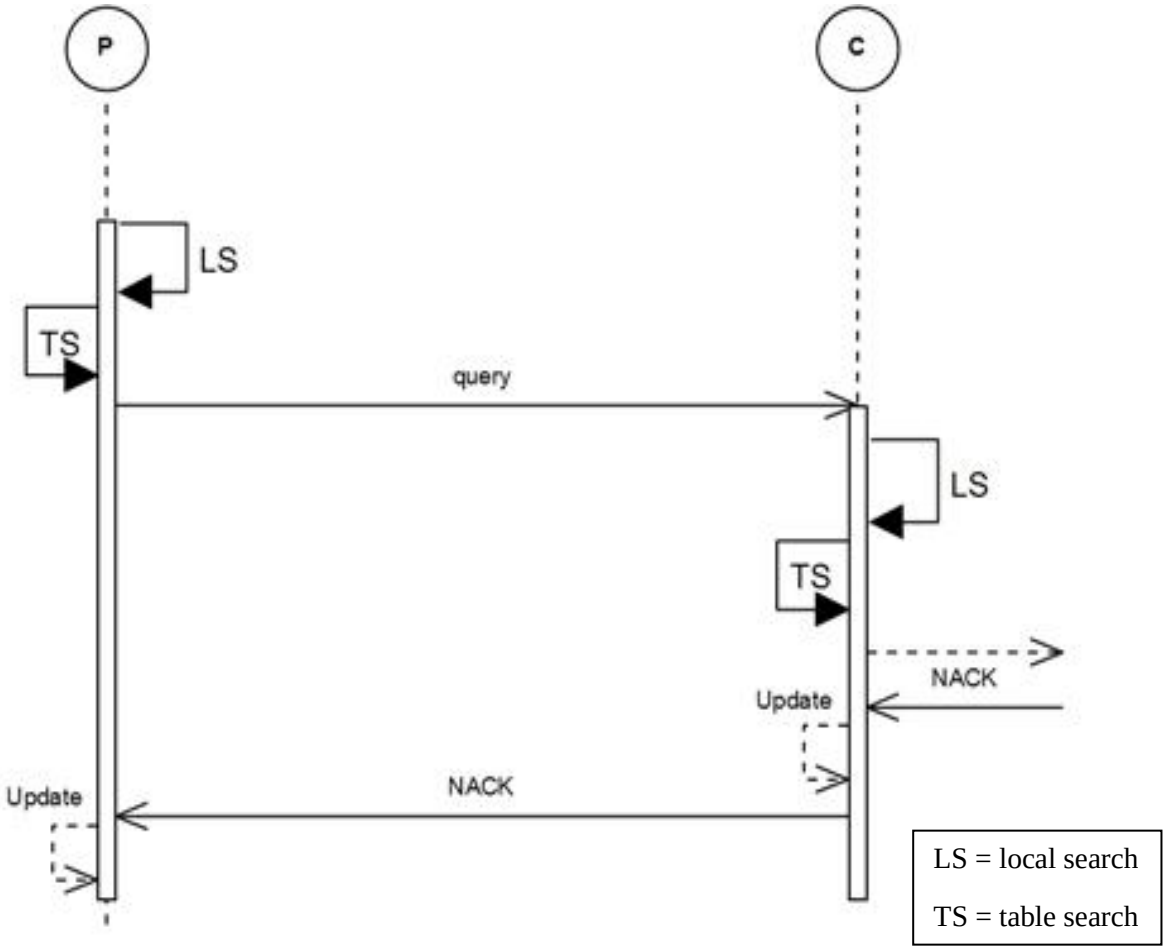


Figure 3. 4 Sequence diagram of search technique

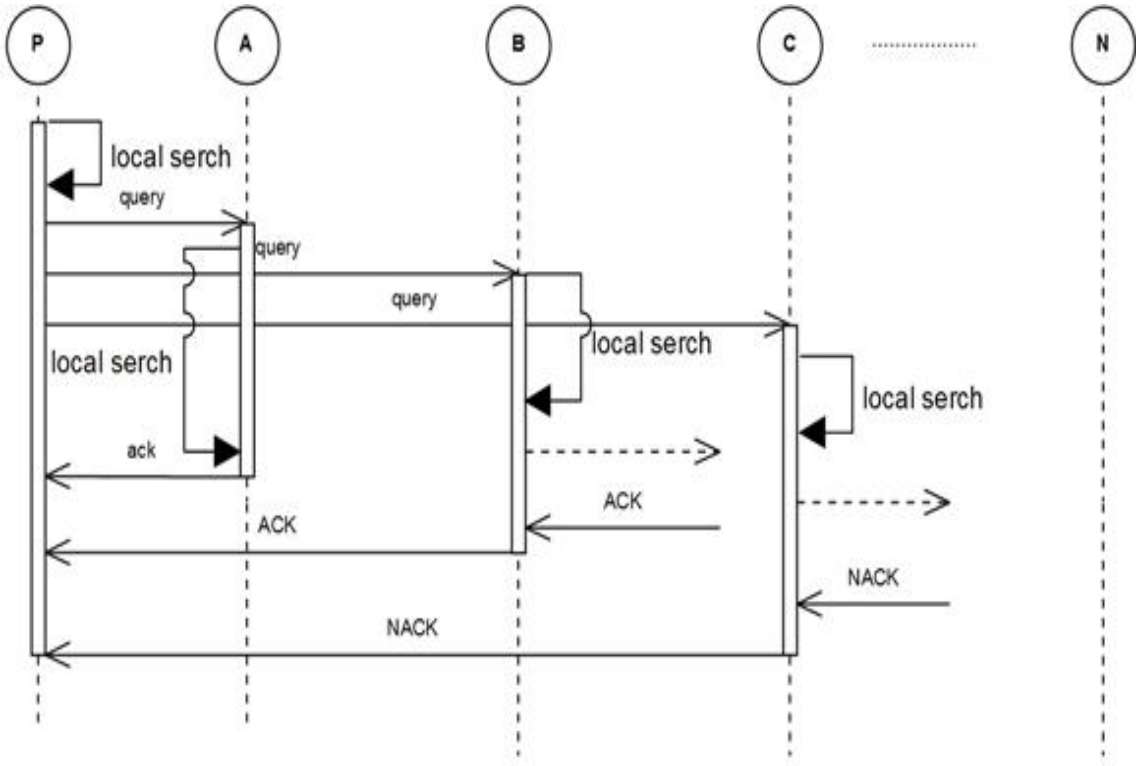


Figure 3. 5 Sequence diagram of Messages exchanged.

Fig3.4 ,Fig 3.5 represents sequence diagram of different search technique and haw the update process work Also various messages sent . As we see There are two processes LS (local search) it means search in local files for this node and TS (table search) it means use the routing table for 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 work when receiving NACK.

2.4. Algorithm:

RIKRW is composed of two core methods, query rotting and update table construction/maintenance, algorithm 1 illustrates the general procedures of the two methods.

- TTL, x , action: statics variable
- Tab_Of_Resultat: It combines the node that contains the desired file
- RT.add: add new routing table
- RT.Calcul_k: calculate the number of walkers 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

2.4.1 query rotting:

```
Procedure  Route_Query(Q.key , S ){
//>>>>>>>>>>Q.key is a id of  query , S is source peer , RT is the routing table
//>>>>>>>>>>R is random neighbor
{
si(TTL<7)
{
TTL=TTL+1
//local search
Si(Exist( Q.key , S)) {
return 1                }
Sinon {
//random neigbbor search
Si(not RT.Exist( Q.key , S )){
RT.add(Q.key , S )
}
sinon
{
// routing table search
tab_k = RT.Calcul_k(Q.key , S)
i=Tab_K.length()
Tantque ( i <> 0) {
Si (Not Route_Query( Q.key , Tab_K[i] )
{
//remove the neighbor i because not respond
RT.update(Q.key ,S, Tab_K[i], 0 )
}
sinon return 1
i=i-1
}}
K = nieghbor /3
Tantque ( K<> 0){
Si(Route_Query( Q.key , R ) )
//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_procedure
```

Algorithm 1: query rotting of RLKRW

2.4.2 update table :

Procedure RT.update(Q.key , S , N, action){

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

Algorithm 2: update table of RLKRW

3. Implementation

Our work aims to improve research in p2p networks through the application of learning, to get an answer to inquire as early and the lowest number of messages, and this is according to the contents of the inquiry.

For testing our scheme, we were led to choose a simulator. 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 RLKRW (reinforcement learning for K random walker) in PeerSim, as presented in the following paragraphs, and we have implanted also the K random walker scheme to make comparison with. We did not implemented Dierct BFS or APS because of the time limitation.

3.1. Simulateur 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 ID the 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 disered file in the RT.

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

Getting the RT of id node for the id file

```
1. public int RLKRW(Node peer,int protocolID,Query in){
   //////////////////////////
2. int linkableID = FastConfig.getLinkable(protocolID);
3. linkable = (Linkable) peer.getProtocol(linkableID);
   //////////////////////////
4. int nodeId=(int)peer.getID();
5. Random rand = new Random();
   //local search ...
6. int id_file_local= RandomDistributionInitializer.id_file(nodeId);

7. int nodeid_cur=nodeId;
8. in.nodeIdsVisited.add(new NodeIds(nodeId));
9. in.hopCount++;
10.msgcount++;
11.boolean fileFound=false;
12.int disired_file=in.id_file.FileName ;


13.if(id_file_local==in.id_file.FileName ){
14.fileFound=true;
15.}

16.if(fileFound){
17.return nodeId;

   //the TTL in condition inferior a 7
18.else if(in.hopCount >= 7){
19.return -1;
20.}

21.else{
22.if(linkable.degree()!=0){

   calculate the k neighbor for route the qeury
23.int k=linkable.degree()/3;

   //Begin Lookup of routing table of the desired file in
   //the node current if not we add
24.if(RT_exist(disired_file,nodeid_cur)!=1)
                                    i.  {
25.RT_add(disired_file,nodeid_cur);
                                    i.  }
26.else{
   //Begin Lookup of routing table
27.int rout_tab[]=get_RT(disired_file,nodeid_cur);

   // route the query for k node by using the routing table of that file
28.int x=0;
29.for (int i = 0; i < k; i++) {
```

```
30. while(rout_tab[x]== 0 && x+1<rout_tab.length){x++;if(x>10)break;}
31. if(rout_tab[x]!= 0){
32. Node neighbor= Network.get(rout_tab[x]);

33. int res =RLKRW(neighbor,protocolID,in);

34. if(res>0){return 1;}
35. else {
    //update the routing table by replace the lose nieghbor
    // update--

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

37. } } }
38. x++;

39. }//end loop for
40. }
    //in the case of RT lose send the query to random neighbor
41. if(fileFound==false){
42. for(int i= 0; i< linkable.degree(); i++){
43. Node neighborToAsk=linkable.getNeighbor(rand.nextInt(linkable.degree()));
44. int pos=RLKRW(neighborToAsk,protocolID,in);
45. if(pos>0 )
46. {fileFound=true;
    //update the routing table by adding new neighbor
    //update++
47. update_RT(disired_file,nodeid_cur,(int)neighborToAsk.getID(),1);
48. return 1;
49. }}}
50. }//end if

    // the file is found
51. if(fileFound){
52. return 1;
53. }

    // in the case of the node leafe
54. if((linkable.degree() <= 0)){
55. return -100;
56. }

57. return -100;

58. }//end method RLKRW
```

procedure proposed RLKRW

```
public void RT_initialize(Node n) {
            int value_id_file;
            int linkableID = FastConfig.getLinkable(pid);
            linkable = (Linkable) n.getProtocol(linkableID);
            int j=0;

            while(j<linkable.degree())
           {
            value_id_file=file_list[(int)linkable.getNeighbor(j).getID()];
            int i=0; while(i<5)if(rt_local[value_id_file][i]==0)break;
                                  else i++;
            rt_local[value_id_file][i]=(int)linkable.getNeighbor(j).getID();

                j++;
           }
            mat_neig.Matrice=rt_local;
                  RT_graph.set((int)n.getID(), mat_neig);

}
```

Figure 3. 6 initialization of RT for each node.

3.3. Simulation results

3.3.1 Simulation setup:

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 KRW and RLKRW with different start node(RANDOM) but the same Query.

Test 2: Comparison between KRW and RL KRW with different start node (RANDOM) and different Query.

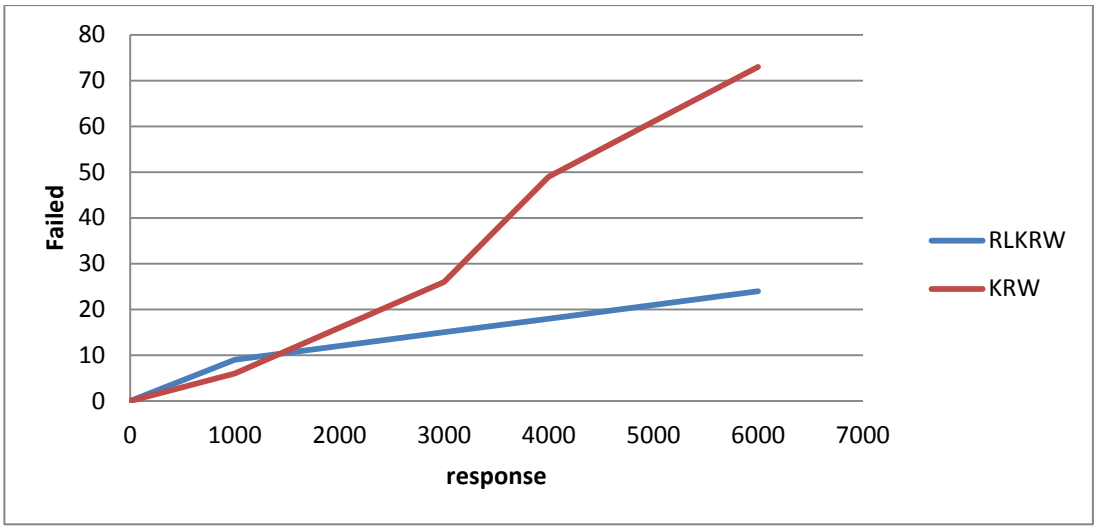


Figure 3. 7 The number of failed operations to get a response (test 1)

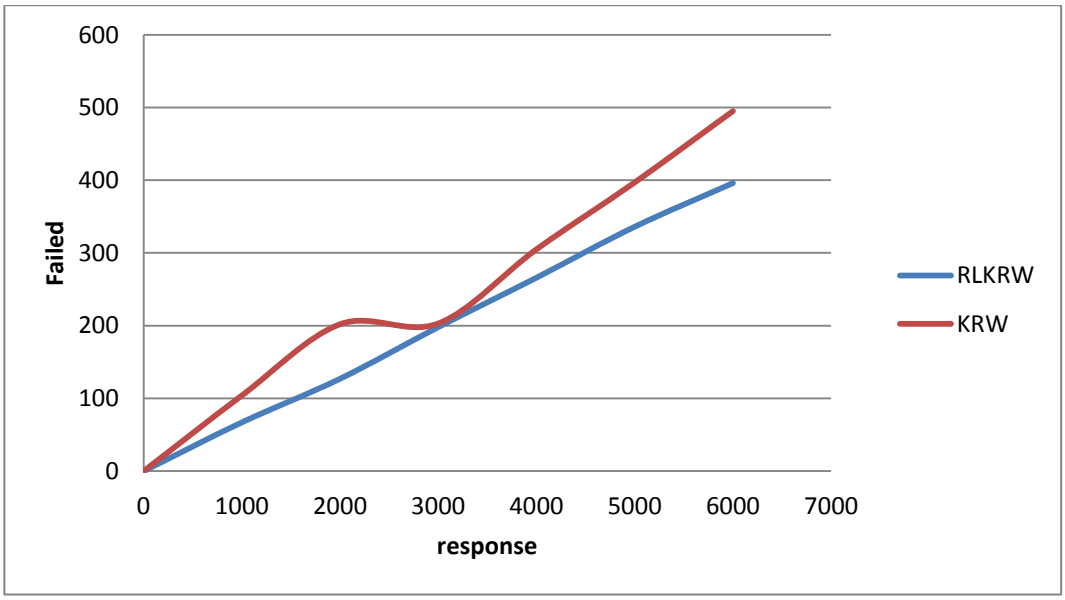


Figure 3. 8 The number of failed operations to get a response (test 2)

Fig 3.7 and **Fig 3.8** Represents a graph of the number of failures that occur until the response gets in the case of different start node(RANDOM) but the same Query or different start node(RANDOM) and different Query Respectively. As we see that the two curves are equal at the beginning, because at the beginning of the search process be random this is because the RT is empty. But after several operations we notice that the number of KRW failures is increasing compared to RLKRW. This is due to random transmitter for a query that is used by KRW Which he abandoned by RLKRW.

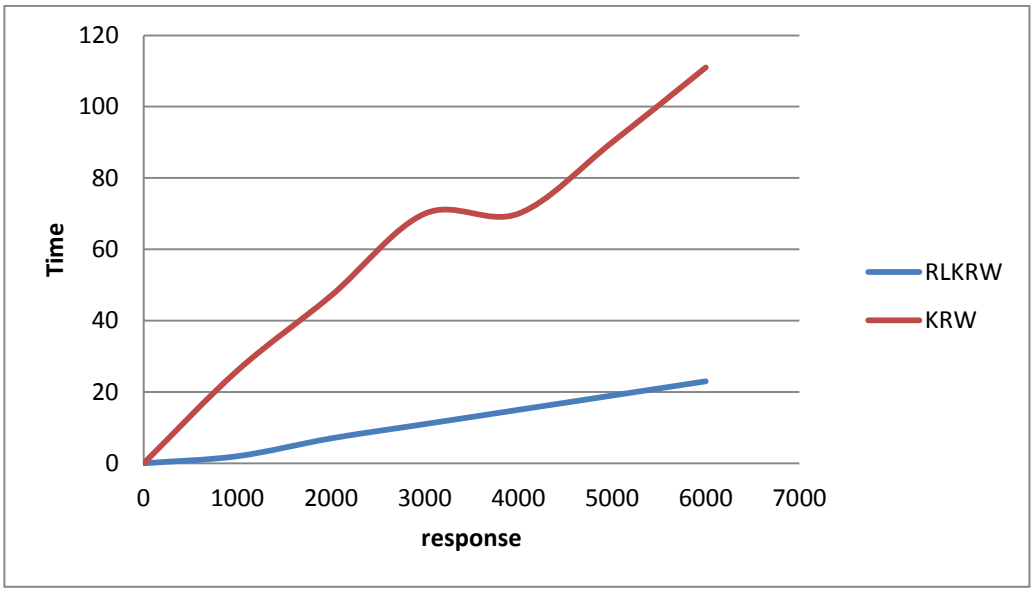


Figure 3. 9 The time it takes to get a response (test 1)

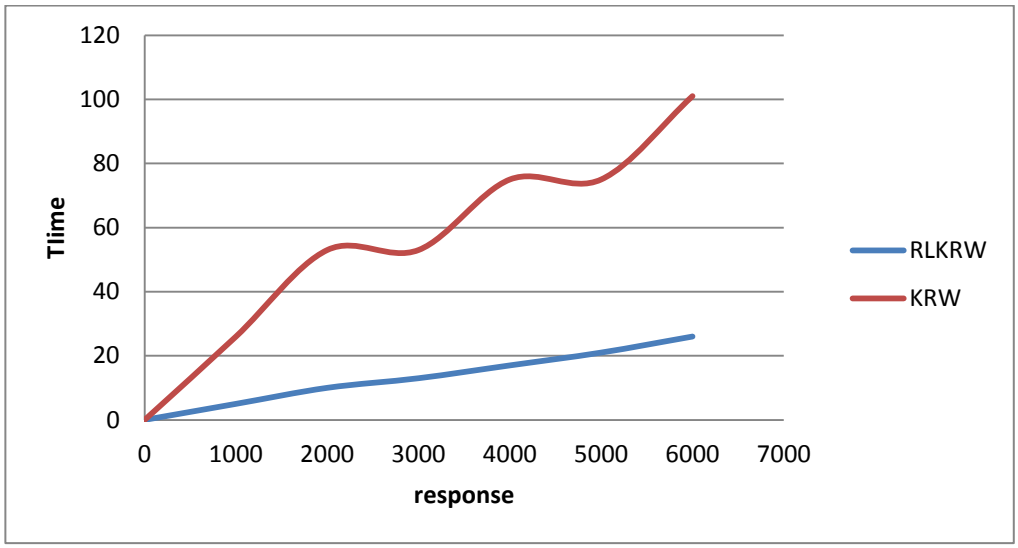


Figure 3. 10 The time it takes to get a response (test 2)

Fig 3.9 and **Fig 3.10** represents a time curve for time taken by both processes KRW and RLKRW 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 KRW, This is due to random transmitter for a query that is used by KRW Which he abandoned by RLKRW.

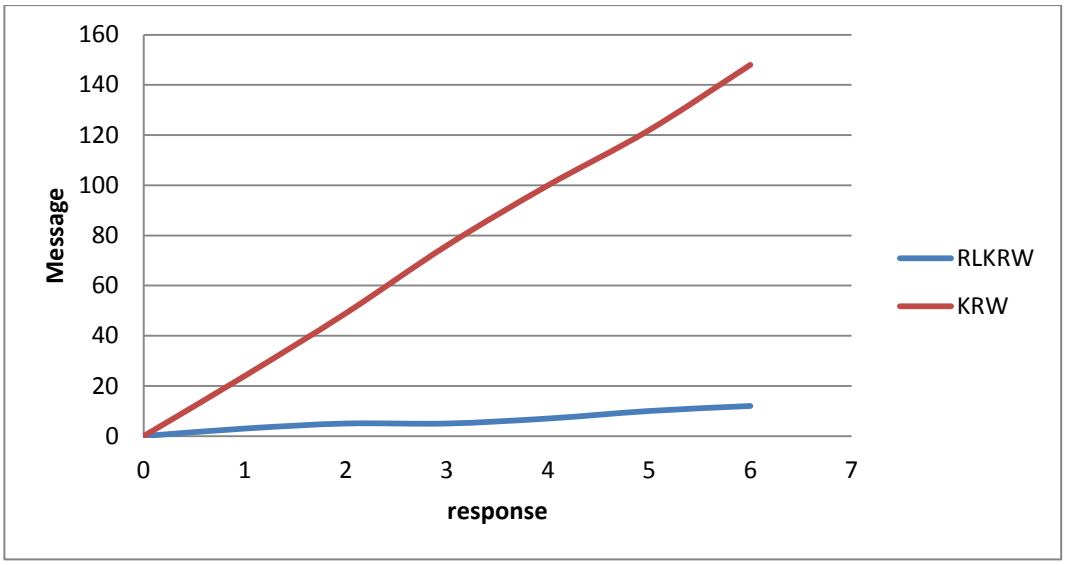


Figure 3. 11 Number of messages sent in order to receive a response (test 1)

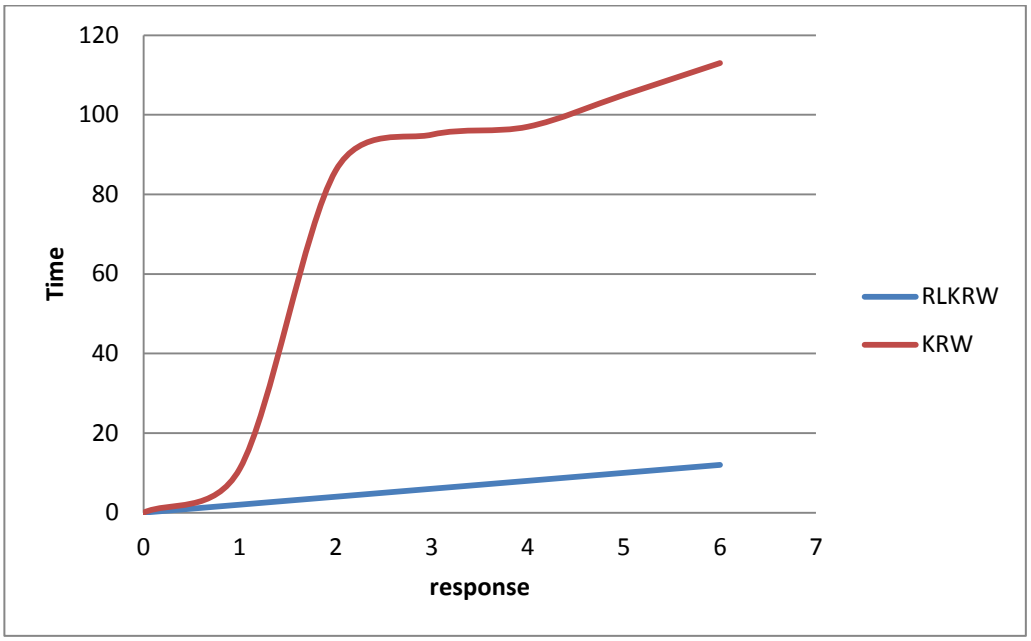


Figure 3. 12 Number of messages sent in order to receive a response (test 2)

Fig 3.11 and **Fig 3.12** in the both case of different start node(RANDOM) but the same Query or different start node(RANDOM) and different Query Respectively. We see RLKRW surpass from the start and that's because it controls the K (the neighbors that sent to) and after few sending of the messages we see stabilization of the RLKRW 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.

3.4 Comparison:

Compared to the k -walker random walk, the RLKRW approach has the same asymptotic performance in terms of the number of failed state . However, by using RT to select neighbor, the RLKRW approach surpasses the k-walker random walk in the query success rate and the message overhead rate (table 3.1).

	Performance of message overhead	query success rate	number of failed state
The k -walker random walk	+	+	+
RLKRW	+ +	+ +	+

Table 3. 1 Compared BETWEEN The K -Walker Random Walk Reinforcement Learning K -Walker Random

RLKRW exhibits many plausible characteristics , such as:

- High accuracy
- Large number of discovered objects
- Robust and adaptive behavior in rapidly-changing environments.

4.Conclusion:

Through the simulation results we found that our proposal perform well then the K random walker (KRW) scheme in exponential manner for the case of the average request response, but it start with the same manner of KRW in beginning and when it learn from the networks about the object it perform well exploiting the traffic change made in the network (fig 3.7, fig3.9).

General Conclusion:

In this work we have interested by the peer to peer networks field, for witch, we have stats by its general describing in the first chatter, followed by the art state of the different information searching techniques in unstructured peer to peer networks that can be divided on two sets: deterministic and probabilistic schemes.

We focus on probabilistic schemes in order to ameliorate its performance by the fact of integrate the traffic change and the object consideration in our proposed scheme RLKRW (Reinforcement Learning K Random Walker).

The simulation results show that our proposal perform well then a KRW (K Random Walker) scheme with exponential manner which encouraged the continuity in this direction.

We propose, as a future work to compare our proposal with the Direct BFS, APS and intelligent search schemes to evaluate more precisely its performance.

Bibliographie :

Books :

[1] C. Watkins and P. Dayan, “Q-learning,” Machine learning, vol. 8, 1992

[2] Xuemin (Sherman) Shen, Heather Yu, John Buford, Mursalin Akon **"** Handbook of Peer-to-Peer **"** Springer Science & Business Media**,** 2010/03/03 Networking

Article:

[3] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for faulttolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, 2001.

[4] 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

[5] L. Gatani, G. L. Re, A. Urso, and S. Gaglio, “Reinforcement learning for P2P searching,” in Proc. of the International Workshop on Computer Architecture for Machine Perception (CAMP’05), 2005.

[6] Li. X, and Wu. J, “Improve Searching by Reinforcement Learning in Unstructured P2Ps,” in 2004 Proc.26th Conf.Distributed Computing Systems ,pp. 75.

[7] Hemant G, Priyanka G, Vikas C “ Different search Algorithms in Unstructured P2P Systems” International Journal of Engineering Technology and Management (IJETM) Available Online at www.ijetm.org

[8] Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content addressable network. In *Proc. the 2003 Conf. Applications, Technologies, Architecture, and Proto- cols for Computer Communications* (*SIGCOMM 2001*), San Diego, USA, August 27-31, 2001, pp.161-172.

[9] Rowstron A, Druschel P. Pastry: Scalable, distributed ob- ject location and routing for large-scale peer-to-peer systems. In *Proc. the 18th IFIP/ACM International Conference on Distributed System Platforms* (*Middleware 2001*), Heidelberg, Germany, November 12-16, 2001, pp.329-350.

[10] Stoica, I., Morris, R., Karger, D., & Kaashoek, M. F. (2001). *Chord: A scalable peer-to-peer lookup service for internet applications.* ACM SIGCOMM Computer Communication Review.

[11] Stoica I, Morris R, Nowell D L, Karger D, Kaashoek M, Dabek F, Balakrishnan H. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 2003, 11(1): 17-32.

[12] TorrentFreak “BitTorrent: The "one third of all Internet traffic" Myth".. 24 January 2007. Archived from the original on 26 March 2014. Retrieved 7 April 2013.

[13] Tsoumakos.D and Roussopoulos.N, “Analysis and Comparison of P2P Search Methods,” in 2006 Proc.1st Int. Conf. Scalable Information Systems (INFOSCALE 2006), Article No. 25.

[14] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-yazti, “A local search mechanism for peer-to-peer networks,” in Proc. of the 11th ACM Conference on Information and Knowledge Management (CIKM’02), 2002.

[15] Vlachou, A., Doulkeridis, C., Norvag, K., & Kotidis, Y . Peer-to-Peer Query Processing over Multidimensional Data. (2012). SpringerBriefs in Computer Science.

[16] Wolf, M. (2002). *Speed!: Understanding and Installing Home Networks* . SAMS Publishing 0-672-32186-6.

[17]Xiuqi Li andJie Wu," Searching Techniques in Peer -to - Peer Networks", a survey of major searching Department of Computer Science and Engineering,Florida Atlantic University 2007

Websites:

[18] . http://www.gnutella.com consulted on 04/04/2016

[19] Wikipedia.Gnutella. https://en.wikipedia.org/wiki/Gnutella consulted on 16/04/2016

ملخص :
شبكة الند للند هي واحدة من المجالات الحديثة والتي تأخذ مكانا هاما في مجال البحث في شبكات المعلوماتية. ويمكن تصنيف برامج البحث الموجودة في P2Ps غير الهيكلية إما القطعية أو الاحتمالية. جودة نتائج الاستعلام في المخططات الحتمية منخفضة. المخططات الاحتمالية تستخدم الاستدلال البسيط الذي يفتقر إلى خلفية النظرية لدعم نتائج أكثر دقة وبشكل جيد. في هذه الوثيقة، نقترح تحسين البحث عن طريق تعزيز التعلم (RL)، والتي أثبتت في الذكاء الاصطناعي انها قادرة على تعلم أفضل سلسلة من الإجراءات من أجل تحقيق هدف معين. مقاربتنا، RLKRW، يهدف إلى تحديد أفضل مسار إلى الملفات المطلوبة من خلال استغلال تغيير حركة البيانات. ويستكشف سبلا جديدة عن طريق إعادة توجيه الاستفسارات إلى K جيران التي تم اختيارها عشوائيا. فهو يستغل أيضا المسارات التي تم اكتشافها لخفض تكلفة الاستعلام التراكمية. نتيجته التجريبية تدعم تحسين أداء .RLKRW
الكلمات المفتاحية: شبكة الند للند ، وتعزيز التعلم، RLKRW ، P2Ps غير منظم.

Abstract:
Peer to Peer network is one of the ambient and newly fields that take an important place in informatics networking search fields. Existing searching schemes in unstructured P2Ps can be categorized as either deterministic or probabilistic. The quality of query results in deterministic schemes is low. Probabilistic schemes use simple heuristics that lack the theoretical background to support more accurate and well results. In this thesis, we propose to improve searching by reinforcement learning (RL), which has been proven in artificial intelligence to be able to learn the best sequence of actions in order to achieve a certain goal. Our approach, RLKRW (reinforcement learning K random walker), aims at locating the best path to desired files by exploiting the traffic change. It explores new paths by forwarding queries to K randomly chosen neighbors. It also exploits the paths that have been discovered to reduce the cumulative query cost. Its experimental result supports the performance improvement of RLKRW.
Keywords: Peer to Peer network, reinforcement learning, RLKRW, unstructured P2Ps.

Résumé:
Réseau Peer to Peer est l'un des champs ambiants et nouvellement qui prennent une place importante dans l'informatique en réseau de champs de recherche. Régimes de la recherche existants dans P2Ps non structurées peuvent être classés comme étant soit déterministe ou probabiliste. La qualité des résultats de la requête dans les schémas déterministes est faible. Schémas probabilistes utilisent des heuristiques simples qui manquent le Contexte théorique pour soutenir des résultats plus précis et bien. Dans cet article, nous proposons d'améliorer la recherche par l'apprentissage par renforcement (RL), qui a fait ses preuves dans l'intelligence artificielle pour être en mesure d'apprendre la meilleure séquence d'actions en vue d'atteindre un certain objectif. Notre approche, RLKRW, vise à localiser le meilleur chemin vers les fichiers désirés en exploitant le changement de la circulation. Il explore de nouvelles voies en envoyant des requêtes à K voisins choisis au hasard. Il exploite également les chemins qui ont été découverts pour réduire le coût de la requête cumulative. Son résultat expérimental soutient l'amélioration de la performance des RLKRW.
Mots clés:Réseau Peer to Peer, apprentissage par renforcement, RLKRW, P2Ps non structurée.