Enhancing the P2P protocols to support advanced multi-keyword queries

Samir Ghamri-Doudane[1], Nazim Agoulmine[2]

[1]LIP6-CNRS, University of Paris 6, France
8, rue du Capitaine Scott – 75015 – Paris – France
samir.ghamri-doudane@lip6.fr
[2]LRSM, University of Evry, France
nazim.agoulmine@lip6.fr

Abstract. Recently, Peer-to-Peer has become a popular paradigm for building distributed systems, aiming to provide resource localization and sharing in large-scale networks. However, advanced searching for resources remains an open issue. The flooding technique used by some Peer-to-Peer systems is expensive in bandwidth usage, and it shows a serious lack in scalability. Also, more efficient systems based on distributed hash tables (DHT) lack in query expressiveness and flexibility. This paper addresses this issue by discussing existing solutions, and proposing a novel approach to support advanced multi-keyword queries in the context of Peer-to-Peer systems. It extends the existing, and widely established, DHT-based localization frameworks. This new approach can substantially reduce the bandwidth consumption and improve the load balancing over the network.

Keywords. Peer-to-Peer systems, Distributed Hash Tables, Content discovery, Keyword-based Searching.

1. Introduction

These last years have seen the emergence and the deployment of a new approach for distributed systems known as peer-to-peer. This approach allows the deployment of distributed systems in large-scale and dynamic networks. Nowadays, the main application of this approach is the object localization in these large scale networks. Objects can be of any type, such as files, services, applications, devices, etc. Though they have introduced a new manner to handle the resource discovery problem, the initial systems had some major limitations. One can mention Napster and its central index that introduces a bottleneck in the network and, therefore, a loss of reliability in addition to high administration costs. In the case of the Gnutella system [6], its expensive diffusion principle introduces a serious load in the network, as well as a serious lack in term of scalability.

To improve the efficiency of the classical approaches, distributed Hash Tables (DHT) have been introduced [16] [12] [17] [10]. The objective of the DHT-based peer-to-peer systems is mainly the localization of files (i.e. the node containing the requested file). These localization systems are based on the construction of a simple index containing associations between File IDs and Node IDs, where peers and data are structurally organized. This index is distributed over the network by the mean of a specific hash function. The DHT-based systems guarantee an efficient discovery with a limited number of hops. Furthermore, the study presented in [9] has demonstrated that DHT-based protocols are suitable for dynamic and large environments.

Despite the efficiency of these approaches, objects can only be localized using a unique identifier. Thus, current DHTs are limited to pure lookup of these identifiers. This unique ID introduces a problem as a user is not always aware of its value (i.e. the name of the corresponding file). However, in order to use these systems in the general case of resource discovery, it is necessary to define an enhanced search approach not only based on a unique ID but on several parameters, possibly fuzzy parameters, that can describe this resource (keyword searching). A system based on this principle will provide a query engine allowing this type of requests in large scale networks.

The objective of this work is to propose and evaluate a new solution to advanced multi-keyword search in the context of structured peer-to-peer systems. The related architecture is completely distributed and is based on DHT-localization. However, it is not based on a specific protocol but aims to incorporate, with minimal efforts, several protocols such as Tapestry [17] and Chord [16]. To support keyword-based requests, the proposed solution introduces a query engine on the top of these DHT-based protocols.

The rest of this paper is organized as follows. Section 2 provides and discusses related works. In section 3, we introduce our novel architecture. The sections 4 and 5 detail our propositions as well as the underlying mechanisms. We evaluate the solution in section 6 before we conclude with section 7.

2. Related work and discussion

Some work has already been done to make peer-to-peer keyword searching feasible. Most of the proposed solutions use the concept of reverse hash tables. The association < ID of file, Node > is replaced by the inverted list: < keyword, List of nodes >. Each resource is described by a list of keywords. Then, each keyword is indexed separately. Hence, the inverted index is distributed among peers by keyword. A query with k keywords can be answered by k nodes. Afterwards, all the results are collected by the initiator of the query and the final result is identified as the intersection of all these responses. Despite its simplicity, we can easily notice the overload introduced in the network by this approach, since the final result corresponds only to a small portion of the received responses. Based on this method, several recent propositions and improvements have been proposed. We can cite:

Reynolds and Vehdat [13] have proposed an architecture based on reverse hash tables associated with Bloom filters and caches to reduce network traffic.

Balazinska et Al. [1] have designed a resource discovery system, called *Twine*, based on the Chord [16] localization protocol. In this system, the support of keyword-based queries is achieved by the translation of resource descriptors into hierarchical trees (dependence between resources' attributes). This relation aims to reduce the load during the creation of the reverse hash tables.

Shi et Al. [15] used, also, the concept of reverse hash tables (keyword indexing). However, this mechanism is improved by organizing the nodes into several groups of different levels depending on their locations. This method aims to reduce the query routing latency as well as the network load.

Even so, the "*reverse hash table*" approach introduces a significant load in the network and nodes. In fact, each resource - and then each query - can be represented by potentially a number of keywords. Moreover, this approach raises the problem of "common keyword" which produces a heterogeneous load in the network. Thus, the node responsible for this so called "common keyword" will be requested more frequently than others, and consequently will be overloaded. The scalability limitations of this technique and its existing optimizations, in term of high bandwidth consumption, have been demonstrated in [14].

Other interesting works presented in [7] [11], introduce new concepts and architectures that are not compatible with existing P2P systems which is not the approach we have chosen. In fact, our objective is to exploit and extend the existing peer-to-peer systems, since they are widely used and accepted. Also, the proposed solution should provide a generic framework, which can be used for various purposes and applications such as: service discovery, file sharing, distributed file storage …etc.

Therefore, through this work, we aim to propose a new technique to handle advanced multi-keyword lookup queries in a large scale peer-to-peer environment. Indeed, this new approach intends to tackle the following objectives and requirements:

– **Generic framework:** use and extend the existing, and widely used, peer-to-peer frameworks. Thus, the new approach should be compatible with the current technologies.
– **Bandwidth saving:** reduce the bandwidth consumption to its minimum in order to improve scalability.
– **Load balancing:** reduce the load disparity between peers. However, this latter is not our main goal, and even if our proposed approach can reduce the load unfairness in the network, it still needs the introduction of more specific load balancing techniques [5] [2] in order to be completely efficient.

Hence, after describing our novel approach, we will evaluate its performances and compare it to the existing ones in order to prove these enhancements.

3. Proposed architecture

In our solution, the system is deployed in a distributed manner on top of a set of participating nodes that communicate together using a peer-to-peer protocol. Each node supports the proposed software architecture as presented in Figure 1. The enhanced query layer is responsible for handling application requests and translating them into localization queries. This layer is deployed on the top of a DHT-based protocol such as Tapestry or Chord to take benefit from their routing and resource localization mechanisms. In the following part of this paper, we will present the various query mechanisms, as well as their integration with the content-localization protocol.

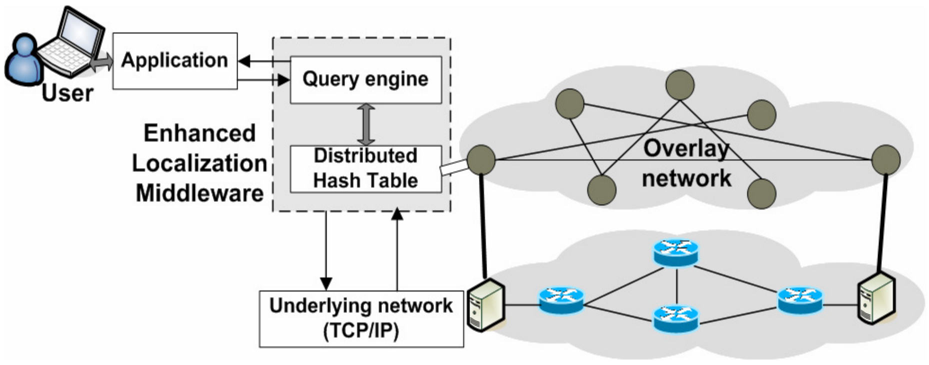


Fig. 1. Software architecture on a node.

4. The query engine

The query engine constitutes the upper part of our discovery system. It receives high level search queries from the client application. Then, it translates them into routable queries which are forwarded to the underlying DHT-based localization layer. Afterwards, it analyses and combines the collected responses before delivering an accurate result to the application. So, the primary purpose of this engine is to provide advanced resource descriptions and a powerful query construction allowing the use and combination of various keywords.

4.1 Identifier Format and resource descriptions

In the current content-localization systems, the resource identifiers are obtained by hashing an attribute or a list of attributes using a consistent function. This list of attributes defines a single key that identifies uniquely the resource. Thus, if a resource is described by a key:

key = (attributeN°1=value1, attributeN°2=value2),

Then, its identifier will be extracted as follows:

ID = h(key), where *h* represents the hash-function.

The naming space should be very large (generally 160 bits) in order to guarantee the identifier uniqueness with a high probability (consistent-hashing characteristic). In this case, the most used hash-function is SHA-1 [3].

The weakness of such localization systems is their incapacity to deal with complex and advanced queries. This comes from the specification of the Ids. In fact, each resource is identified by its unique key, instead of a list of attributes, which limits considerably its description. In order to respond to this lack, we propose to change the identifiers structure. This latter will be decomposed into several fields. For each resource type, a list of major attributes is established (attributes that should appear in the resource ID). This list should be larger than the key-attributes list and should model the resource in an accurate manner. Thus, the identifier is not based any more on the resource key, but on its description (which is as complete as possible). The final identifier is specified according to the format presented in figure 2. This latter provides a comparison between the usual identifier construction technique and the proposed method.

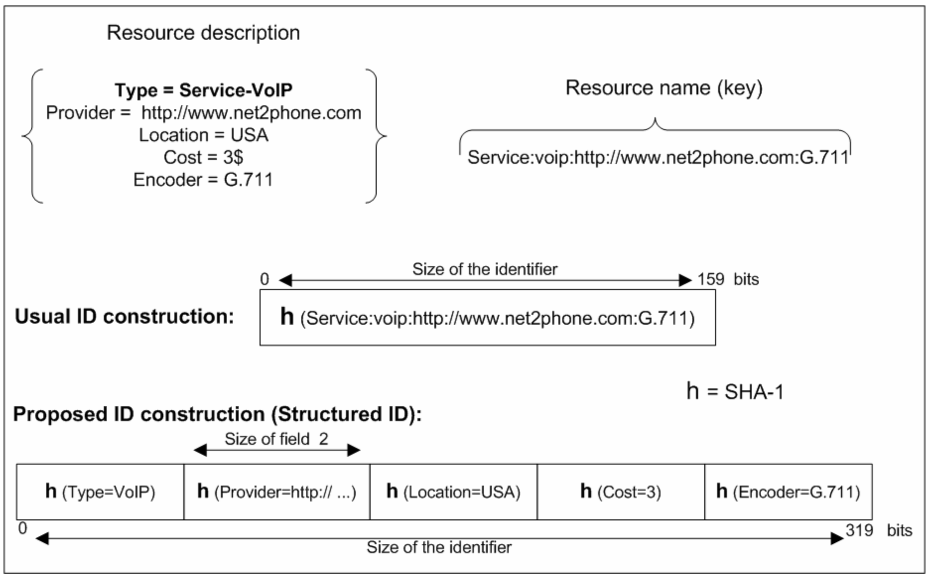


Fig. 2. Identifier format, usual vs. proposed method.

The hash-function, that is used for each couple (attribute = value), is SHA-1. The first attribute should be the ‘resource type’, because it is the one that infers the exact identifier structure. In fact, the number of fields composing an identifier may vary according to the resource type (the list of major attributes describing a resource). Also, the sizes of the different fields can be different in a same ID. The unique constraint is to choose each field size proportionally to the Base *B* of the underlying localization protocol. This constraint aims to facilitate the routing process. Therefore, it is easy to combine the number and the size of the fields in order to comply with this constraint. For example, if a resource is described by 7 attributes and if the total identifier size is 320 bits, one solution is to fix the size of the first field to 80 bits and fix all the remainder to 40 bits. Naturally, the total ID size is the same for all the resources. This size should be very large in order to guarantee uniqueness with a high probability.

Concerning the node identifiers, we keep the same construction technique as in the existing systems, i.e. hashing of the IP address, the public key or any other unique attribute of the node.

4.2 Query construction

Traditionally, in the content-localization systems, the requests are built by specifying the unique key of the desired resource (its identifier). Then, this request is routed to the node indexing this ID. The localization process is completed.

In our discovery system, the user application provides a set of keywords to the underlying query engine in order to formulate its request. Generally, these keywords are only a subset of all the attributes identifying the resource. Hence, the query engine can not, from this subset of attributes, recover a unique identifier to locate a resource. However, it can calculate some of its fields to build a fuzzy identifier (a set or interval of identifiers). Then, the localization protocol diffuses the request to all the nodes indexing the elements of this set of IDs. It is the concept of "limited diffusion". Figure 3 provides an example of such a query construction, based on our "*Fuzzy-identifier*" concept.

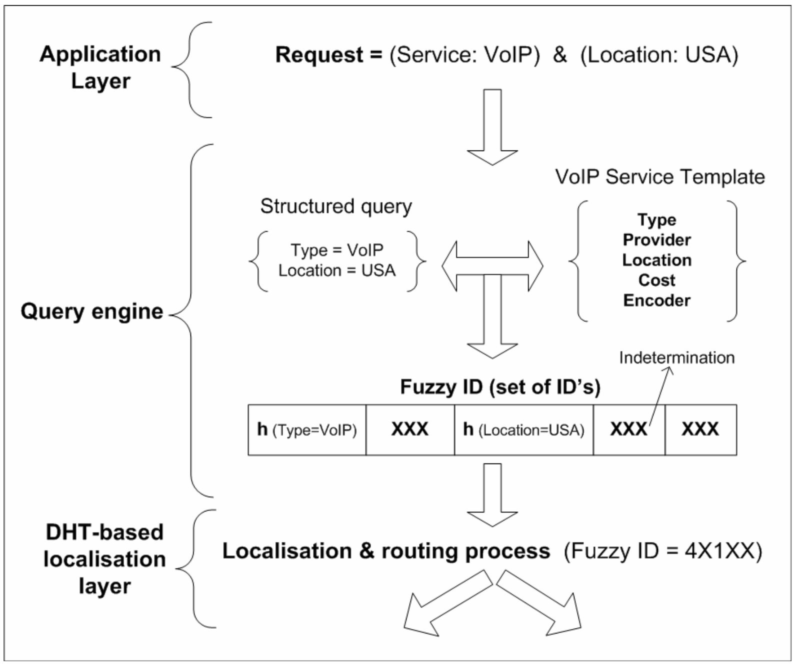


Fig. 3. Query construction example: the "Fuzzy-identifier" concept.

When the user request combines several logical operators: *OR* / *AND*, the query engine translates them according to a canonical template:

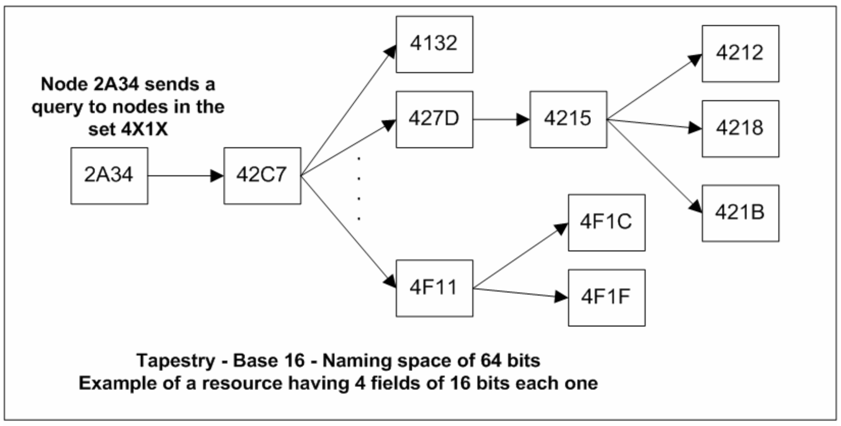
$$(\mathbf{a} \,\&\, \mathbf{b} \,\&\, \mathbf{c}) \;||\; (\mathbf{d} \,\&\, \mathbf{e} \,\&\, \mathbf{f} \,\&\, \mathbf{g}) \;||\; \dots\; (\mathbf{z} \,\&\, \mathbf{f}).$$

Where { **a**, **b**, …, **z** } represents query axioms. For example:

a = "Type=VoIP", **b** = "Location=USA", **c** = "Encoder=G.711", ...

Then, the query is divided into several basic queries according to the union operator. A basic query should contain only intersection operators. Then, each basic query is translated into a fuzzy identifier (an interval or a set of IDs). Thus, each fuzzy identifier constitutes a query which is sent to the localization layer in order to be routed. After receiving the responses, the query engine collects these results, combines and analyses them in order to extract the final result.

5. Query routing: "limited diffusion"


Node 2A34 sends a query to nodes in the set 4X1X
2A34
42C7
4132
427D
4215
4212
4218
421B
4F11
4F1C
4F1F
Tapestry - Base 16 - Naming space of 64 bits
Example of a resource having 4 fields of 16 bits each one

```
NextHop(n,ID) /* return a node or a list of nodes */
if (n = MAX_ID_LENGTH) then
  return self
endif
d <- ID[n]
if (d = XXX) then
  listOfNodes L <- EMPTY_LIST
  i <- 0
  while (i < MAX_DIGIT) do
      e <- R[n,i]
      if (e <> nil) then
        if (e = self) then
          L.add(NextHop(n+1,ID))
        else
          L.add(e)
        endif
      endif
      i <- i + 1
  endwhile
  return L
else
  e <- R[n,d]
  if (e = self) then
    return NextHop(n+1,ID)
  else
    return e
  endif
endif
```

NextHop: this function computes the next hop(s) to locate the given fuzzy ID.
ID: the fuzzy Identifier to locate.
n: the hop number (routing step number).
self: the local node.
R: routing table at the local node.
XXX: indicates an indetermination digit.

Pseudocode of the routing algorithm (simplified version)

Fig. 4. Limited diffusion with Tapestry.

When a resource is published and indexed in the network, its identifier is calculated using its major attributes. Then, the routing layer forwards this ID using the habitual process. However, during the search phase, the query engine provides to this localization layer a set of IDs. At that time, the "limited diffusion" mechanism is solicited, since the requesting node should contact several nodes simultaneously. In figure 4, we describe, as an example, the extension of the Tapestry protocol with this diffusion mechanism.

In the Tapestry protocol [17], every node and every resource is assigned a unique ID. Indeed, the identification space is structured in a hierarchical tree. Thus, the routing is performed in a recursive manner by progressing digit by digit in the targeted node ID, using the routing table entries of each visited node. In the case of our extension, the routing algorithm handles the "*undetermined digits*" by forwarding the request to all the relevant entries in the routing table, as shown in figure 4. Hence, it is a *hybrid* solution (routing vs. diffusion). This extension introduces only minor changes into the functioning (routing algorithm [17]) of this localization protocol. All the other features and algorithms of the Tapestry protocol are kept unchanged, especially in term of replication, fault tolerance, network construction and maintenance. In the same way, our architecture can use and extend other DHT-based localization protocols, such as Chord [16] or Pastry [19].

Also, the localization protocol keeps his performances unchanged. In order to confirm this assertion, we initiated a set of tests, using the p2psim software [4], and concerning our extension of the tapestry protocol. The simulation consists on varying the size '*N'* of the network and the base '*B*' of the protocol (the base of the identification space). The curves in figure 5 exhibit clearly the O(Log$_B$ N) characteristic of the query routing algorithm, in term of mean hop count (the number of hops necessary for a query to reach all its destinations).

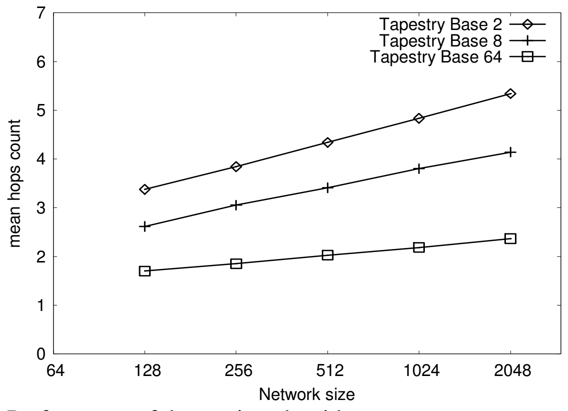


Fig. 5. Scalability Performance of the routing algorithm.

6. Evaluation of the Query engine

In this section, we evaluate the performances of the proposed query engine by comparing our approach, the *fuzzy identifiers* technique, to the *reverse hash table* approach. This latter is described in section 2: Related work. This comparison is mainly based on simulations and made with respect to three crucial features:

- **Storage cost**: by computing the mean index size per node.
- **Load balancing**: by tackling the load disparity between nodes in term of index distribution.
- **Bandwidth cost**: by computing the mean number of messages received in response to a search query.

Before going further in this evaluation, we define in Table 1 a set of experiment parameters and metrics, as a basic *vocabulary* for the rest of this section. Thus, the system has *N* peers. The overall index is composed of *M* resource descriptions and distributed across the network peers. Each resource is described using *K* relevant keywords.

Table 1. Parameters and metrics

Name	Description
N	Number of nodes in the network.
M	Number of documents (resource descriptions) in the network.
K	Mean number of keywords to describe a resource, or document.
r	Replication factor (index replication for availability purposes).
D_i	Size of the index maintained by $node_i$.
D	Mean index size per node.
L	Load disparity factor (in term of index distribution)

The mean index size per node determines the storage cost to distribute and maintain the index over the network peers. This metric is obvious and easy to compute in both cases:

Fuzzy identifiers: $$D = \frac{r \times M}{N} \quad (1)$$

Reverse hash table: $$D = \frac{r \times K \times M}{N} \quad (2)$$

Indeed, in our approach, each resource is described using only one index entry. On the other hand, in the reverse hash table approach, each keyword of the resource is indexed separately (the K factor in the second formula). We can notice the big storage loss due to the use of reverse hash tables in comparison to our approach, since the mean number of keywords per resource is generally big enough.

Another critical issue of the index distribution performance is the load balancing (index balancing over the network peers). In order to study this feature, we defined a specific and accurate metric. This metric is computed as the *standard deviation* of the discrete quantitative variable: $(S_i = D_i / D)$. By definition, the mean value of S_i is 1. Also, the mean index size has no effect on this variable, and thus on its standard deviation. Therefore, this metric reflects only the index load disparity between nodes. In an ideally balanced system, the metric should be equal to zero. Also, high values imply an unfair index distribution as well as the presence of extremely loaded nodes. We call this metric *"Load disparity factor"*:

$$L = \sqrt{\frac{1}{N} \times \sum_{i=1}^{N} \left(\frac{D_i}{D} - 1\right)^2} \qquad (3)$$

Figure 6 compares our approach to the reverse hash table technique. In these experiments, the overall number of documents, *M*, is set to one million, and the mean number of keywords, *K*, is set to 8. The index is not replicated (*r* = 1). First, we define the global keyword and document space, reflecting a realistic situation. Then, inside this space, we chose randomly the set of *M* documents to constitute our index. This latter is distributed across the network hosts according to both techniques. The curve in figure 6 shows the effect of the network size on the load balancing metric. The use of a reverse hash table leads to a highly unbalanced index distribution. Furthermore, the network size has a big effect on the load disparity parameter, which is very constraining for a potentially large scale localization system. On the other hand, our approach showed low and quite stable results. Thus, it is more suitable for a deployment in large scale environments. However, these results are not optimal. The system needs the introduction of more specific load balancing techniques [5] [2] in order to improve its performances.

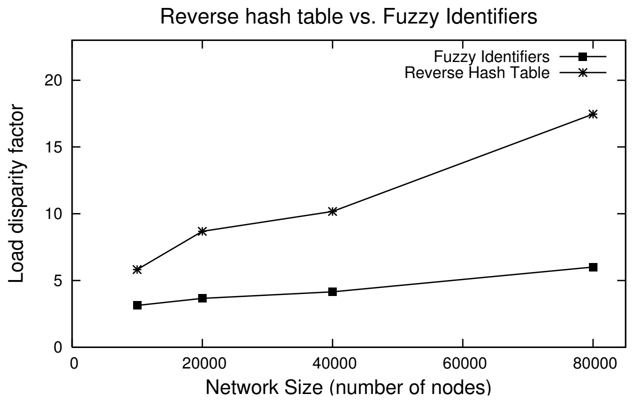


Fig. 6. Load disparity between nodes: the network size effect.

The number of documents received as a response to a search query is a critical issue of a scalable P2P localization system, since it has a big impact on both, the aggregate bandwidth cost and the processing load on peers. This critical parameter is plotted in figure 7 as a function of the query accuracy. By this latter, we mean the number of specified keywords in the query. If a query is more precise, by providing several keywords, it should produce fewer responses. Hence, the mean number of received documents should be inversely proportional to the query accuracy. This assertion is verified in the case of our approach. Furthermore, this approach is optimal since the number of received responses corresponds to the final result of a query; the bandwidth consumption is reduced to its minimum.

In the case of the reverse hash table approach, each keyword is indexed separately; so, a query with *k* keywords is divided into k requests and answered by *k* nodes. Then, all the results are collected and the final result is the intersection of all the received responses. Therefore, the mean number of received documents is proportional to the query accuracy, while the final result is inversely proportional to this parameter, which is awkward. Moreover, as shown by figure 7, the use of a reverse hash table leads to a huge waste of bandwidth. The use of Bloom filters and cache methods [13] may reduce this loss; but it is only a partial enhancement and not a complete solution to this problem.

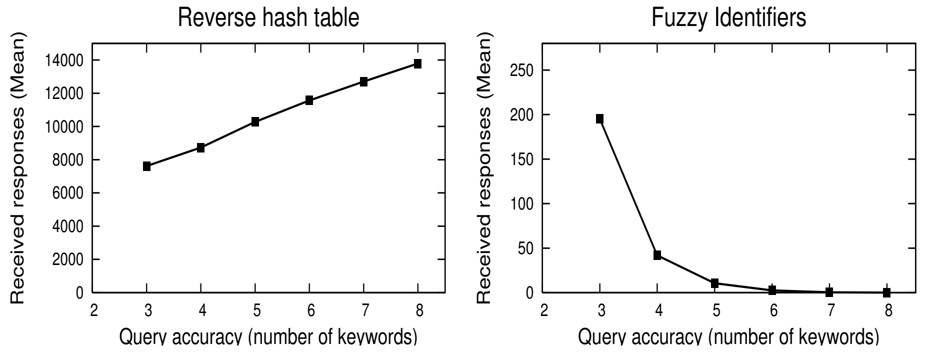


Fig. 7. Bandwidth cost per query: the query accuracy effect.

This waste of bandwidth appears again during the registration phase. In fact, when a new resource is indexed in the system, a registration message is sent for each of its keywords. While, in our approach, only one message is produced and sent during the registration phase, which is more efficient.

10. Conclusion

Advanced keyword searching is not feasible in large scale networks using the usual *reverse hash table* approach, because of its various limitations. For that reason, we have proposed and detailed a new solution to handle this important requirement. This solution exhibits the following strengths: it extends the efficient DHT-based peer-to-peer frameworks, in a generic way; it minimizes the bandwidth and storage costs; and it reduces the unfairness related to the index distribution. All these assertions have been demonstrated through various simulations and experiments. Despite these improvements, the index distribution problem is still open and therefore we are investigating an efficient load balancing algorithm in order to improve.

As a future work, we will use our enhanced localization system in a global service discovery architecture for large ambient networks, where the number of resources can be extremely large and dynamic. This architecture should overlap the existing local service discovery systems.

11. References

1. Balazinska M., Balakrishnan H., Karger D., « INS/Twine : A scalable peer-to-peer architecture for intentional resource discovery», *Pervasive 2002 - 1st International conference on Pervasive computing,* Zurich, Switzerland, 26-28 August 2002.
2. Byers J., Considine J., Mitzenmacher., « Simple load balancing for distributed hash tables », *Proceedings of the 2nd IPTPS*, Berkeley, CA, USA, 20-21 February 2003.
3. FIPS 180-1., « Secure hash standard » U.S. Department of Commerce/NIST, Springfield, VA, April 1995.
4. p2psim, ***http://pdos.lcs.mit.edu/p2psim/***.
5. Godfrey B., Lakshminarayanan K., Surana S., Karp R., Stoica I., « Load balancing in dynamic structured P2P systems », *Proceedings of IEEE INFOCOM 2004*, Hong Kong, 7-11 March 2004.
6. Gnutella, ***http://gnutella.wego.com/*** .
7. Heng Tao Shen, Yanfeng Shu, Bei Yu, «Efficient Semantic-Based Content Search in P2P Network» *IEEE Transaction on Knowledge and Data Engineering*, No. 7, July 2004, pp. 813-826.
8. Harren M., Hallerstein M., Huebsch R., Thau B., Shenker S., Stoica I., « Complex queries in DHT-based peer-to-peer networks », *Proceedings of the 1st IPTPS*, Cambridge, MA, USA, 7-8 March 2002.
9. Li J., Stribling J., Kaashoek F., Morris R., Gil T., « A performance vs. cost framework for evaluating DHT design tradeoffs under churn », *Proceedings of IEEE INFOCOM 2005*, Miami, FL, USA, 13-17 March 2005.
10. Meymounkov P., Mazieres D., « Kademilia : A peer-to-peer information system based the XOR metric », *Proceedings of the 1st IPTPS*, Cambridge, MA, USA, 7-8 March 2002.
11. Mischke J., Stiller B., « Rich and Scalable Peer-to-Peer Search with SHARK », *Autonomic Computing Workshop, Fifth Annual International Workshop on Active Middleware Services (AMS'03)*, 25 June 2003.
12. Ratnasamy S., Francis P., Handley M., Karp R., Shenker S., « A scalable content-addressable network », *Proceedings of SIGCOMM 2001*, San Diego, CA, USA, August 2001, p. 161-172.
13. Reynolds P., Vahdat A., « Efficient peer-to-peer keyword searching », *Proceedings of ACM/IFIP/USENIX Middleware 2003*, Rio De Janeiro, Brazil, 16-20 June 2003.
14. Jinyang Li, Boon Thau Loo, « On the Feasibility of Peer-to-Peer Web Indexing and Search », *Proceedings of the 2nd IPTPS*, Berkeley, CA, USA, 20-21 February 2003.
15. Shi S., Yang G., Wang D., Yu J., Qu S., « Making peer-to-peer searching feasible using multi-level partitioning », *Proceedings of the 3rd IPTPS*, San Diego, CA, USA, 26-27 February 2004.
16. Stoica I., Morris R., Karger D., Kaashoek F., Balakrishnan H., « Chord : A scalable peer to peer lookup service for internet applications », *Proceedings of SIGCOMM 2001*, San Diego, CA, USA, August 2001, p. 149-160.
17. Zhao B., Huang L., Stribling J., Rhea S., Joseph D., « Tapestry: A Resilient Global-Scale Overlay for Service Deployment », *IEEE Journal on Selected Areas in Communications*, vol. 22, n° 1, January 2004, p. 41-53.
18. Rodrigues R., Liskov B., « High availability in DHTs: Erasure coding vs. Replication », *Proceedings of the 4th IPTPS*, Ithaca, NY, USA, 24-25 February 2005.
19. Rowstron A., Druschel P., « Pastry: Scalable, distributed object location and routing for largescale peer-to-peer systems », *Proceedings of ACM/IFIP Middleware 2001*, Heidelberg, Germany, 12-16 November 2001.