Load Reduction in the KAD Peer-to-Peer System

Moritz Steiner[1,2], Wolfgang Effelsberg[2], Taoufik En-Najjary[1], and Ernst Biersack[1]

[1] Institut Eurecom, Sophia–Antipolis, France
steiner,ennajjar,erbi@eurecom.fr
[2] University of Mannheim Germany
effelsberg,steiner@informatik.uni-mannheim.de

Abstract. Distributed hash tables (DHTs) have been actively studied in literature and many different proposals have been made on how to organize peers in a DHT. However, very few DHTs have been implemented in real systems and deployed on a large scale. One exception is KAD, a DHT based on Kademlia, which is part of eDonkey, a peer-to-peer file sharing system with several million simultaneous users. In this paper, we investigate the publishing and searching mechanisms in KAD. We designed and implemented *Mistral*, a content spy that can capture up to ten million references to published content in several hours. At first evaluation, we notice that publishing new content in a KAD system is much more expensive than searching and retrieving existing content. Indeed, measurements show that of all the Internet traffic generated by KAD-based peer-to-peer networks, 90% is for publishing and 10% for retrieving existing files. Moreover, the most frequently published keywords are meaningless stopwords. We propose to add a stopword filtering mechanism to the search and publish procedures of KAD-based peer-to-peer systems.

1 Introduction

In recent years, Internet traffic from peer-to-peer applications has by far exceeded all other kinds of Internet traffic, including WWW access [8]. KAD-based peer-to-peer systems have become very popular, counting several millions users [19]. Thus, it is important to investigate the performance of KAD-based search and publishing, and to suggest improvements to the KAD mechanisms.

KAD is a Kademlia-based peer-to-peer routing system. Kademlia [10] is a distributed hash table that is implemented in several popular peer-to-peer applications, such as Overnet [12], eMule [7] and aMule [1]. Each KAD node has a global identifier, referred to as KAD ID, which is 128 bit long and is randomly generated using a cryptographic hash function. Routing in KAD is based on prefix matching using a XOR-metric.

The KAD system is designed to prevent free-riding, anyone who retrieves a file from KAD also becomes a server for that file, and he publishes this fact to the rest of the world. Thus, new publications are a consequence of successful retrievals. To investigate the publishing process of KAD, we designed and implemented our own content spy, *Mistral*, that will be described in section 3.

We first launched Mistral on the entire KAD ID space. The load on our machine was too large, and not all the queries could be satisfied and recorded. Since spying on the entire KAD ID space was not possible, we spied on 20 different 8-bit zones (such a zone contains the peers having the 8 first bit in common) of the KAD ID space during 24 hours[3], which allowed us to obtain a number of original results. We observed for a single 8-bit zone :

- More than 1.5 million different references to published files,
- More than 40,000 different keywords were published and only 1,100 searched,
- Publishing generates ten times more messages than searching. Moreover, publish messages are ten times larger than publish messages.
- The popularity of the keywords is not at all uniformly distributed, as it was also observed for other DHTs [3, 17].
- The most popular keywords are meaningless stopwords.

These results led us to look into means how to effectively reduce the publishing overhead without reducing the retrieval success rate of the KAD system. From the indexing and retrieval literature it is well known that eliminating *stopwords* helps to reduce unnecessary searches. Stopwords are very common words of a language that do not contribute much to the power of an index; examples are "the", "a" or "what" that occur frequently without adding much meaning. When a full-text index is built, these stopwords are usually excluded. For example, in the English language, 26 stop words make up 33% of a text.

All DHT-based peer-to-peer systems index files based on the file names, and so does KAD. However, KAD's only mechanism related to stopwords is to leave out all one- and two-letter words when creating an index for a file name. This does not seem to be an efficient solution. As a consequence, we decided to

- Measure the performance of a KAD-based peer-to-peer system with *Mistral*,
- Provide a specific set of stopwords and exclude these stopwords from the index creation and search process,

Our paper is structured as follows. Section 2 presents a short overview of the inner workings of KAD. In Section 3, we introduce our KAD content spy. In Section 4 we analyze the performance of KAD without and with stopwords. In Section 5 we discuss related work on stop words in peer-to-peer systems. Section 6 concludes the paper.

2 Background on KAD

KAD is a Kademlia-based [10] peer-to-peer DHT routing protocol implemented by several peer-to-peer applications such as Overnet [12], eMule [7], and aMule [1].

[3] This is the minimum observation time required to catch also rare keywords, since the typical keyword republishing interval is 24 hours.

eMule and aMule, two open–source projects, have the largest number of simultaneously connected users since these clients rely on the eDonkey network, which is a very popular peer-to-peer system for file sharing. Recent versions of these clients implement the KAD protocol.

Similarly to other DHTs like Chord [20], Can [15], or Pastry [16], each KAD node has a global identifier, referred to as KAD ID, which is 128 bit long and is randomly generated using a cryptographic hash function. The KAD ID is generated when the client application is run for the first time and is then permanently stored. The KAD ID stays unchanged on subsequent join and leaves of the peer, until the user deletes the application or its preferences file.

2.1 Routing

Routing in KAD is based on prefix-matching: Node a forwards a query, destined to a node b, to the node in his routing table that has the smallest XOR-distance[4]. The fact that the XOR metric is symmetric is an advantage compared to other systems, e.g. Chord, since in KAD, if a is close to b, then b is also close to a.

The entries in the routing tables are called *contacts* and are organized as an unbalanced *routing tree*: a peer P stores only a few contacts to peers that are far away in the overlay but increasingly more contacts to peers as we get closer P. For details of the implementation see [21]. For a given prefix distance, P knows not only a single peer but a *bucket* of contacts. Each bucket can contain up to ten contacts, in order to better cope with peer churn without the need to periodically check if the contacts are still online.

Routing to a specific KAD ID is done in an iterative way, which means that each peer on the way to the destination returns the coordinate of the next hop to the sending node. While iterative routing experiences a slightly higher delay than recursive routing, it offers increased robustness against message loss. The useful side effect is that it greatly simplifies crawling the KAD network.

2.2 Publishing

A **key** in a peer-to-peer system is an identifier used to retrieve information. KAD distinguishes between two different keys:

– A **source key** that identifies the content of a file and is computed by hashing the *content* of a file.
– A **keyword key** that classifies the content of a file and is computed by hashing a single token from the *name* of a file.

In KAD each key is not published just on a single peer that is numerically closest to that key, but on 10 different peers whose KAD ID agrees at least in the first 8-bits with the key. This zone around a key is called the *tolerance zone*.

[4] The XOR-distance $d(a, b)$ between nodes a and b is $d(a, b) = a \oplus b$. It is calculated bitwise on the KAD IDs of the two nodes, e.g. the distance between $a = 1011$ and $b = 0111$ is $d(a, b) = 1011 \oplus 0111 = 1100$.

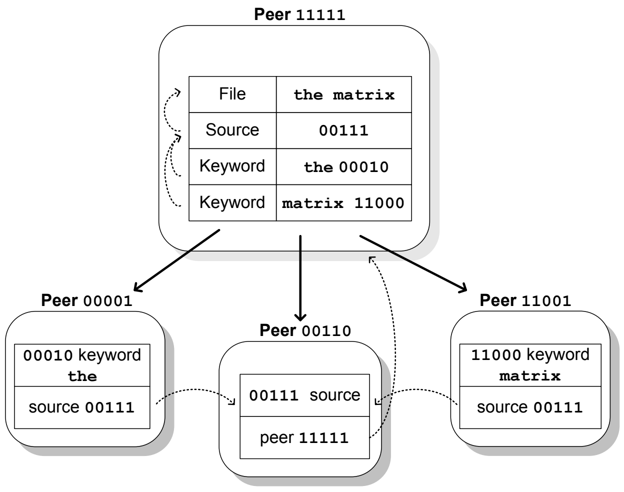

Fig. 1. Sketch of the 2-level publishing scheme

Figure 1 shows an example of the publishing process. A peer wants to publish a file with the name `the_matrix`. This filename will result in two keywords, "the" and "matrix". All relevant references to the original file are generated, such as the source key and the the keywords with the attached metadata. Next, the keywords "the" and "matrix" are published, pointing to the source key. Finally, the source is published, with pointer to the publishing peer.

Keys are periodically republished: **source keys** every 5 hours and, **keyword keys** every 24 hours. Analogously, a peer on which a source key or keyword key was published will delete the information after 5 and 24 hours respectively. Republishing is done exactly the same way as publishing.

3 Spying for Content with *Mistral*

In this section we explain how to get an overview of the content in KAD using our content spy *Mistral*. It is based on the same principle as the *sybil attack* [6, 5, 4]. We introduce a large number of our own peers, the *sybil*s, into the network, all controlled by one machine. Positioned in a strategic way in the KAD space but physically all running on the same machine, the sybils can gain control over a fraction of the network or even over the whole network. The fact that all sybils run on the same machine has the advantage that data collection is much easier.

We insert a large number of sybil peers into the network and propagate information about them in the routing tables of the real peers. To do so, we first crawl KAD using *Blizzard* [19] to learn about the peers present in the network. Next, we send `hello` messages to the peers we have learned about. A `hello` message includes the KAD ID of the sender; we can choose it freely. The first 24 bit are chosen at random, while the 96 remaining bits are fixed.

The routing queries reaching the sybils are always answered with other sybils. The returned KAD ID is closer to the target included in the query than the receiver of the query, thus the querying peer has always the impression of approaching the target. Once the requestor gets close enough to the target, it queries a sybil for the content itself and not for any closer peers. Our sybil stores the search request and returns a fake source entry. This source entry points to our machine. As a consequence, the real peer tries to start to download which is not successful.

Not only routing and search requests are hitting our machine but also requests to publish. As stated above, these are especially interesting since they are much more frequent than search requests. Whereas search requests are always launched by a human, publish requests are automatically and regularly launched by the KAD clients. Also, the publish information is richer than the search requests: it includes the full file name, the KAD ID of the source and a significant amount of metadata on the file. As explained above, the filename is tokenized and published on the hash of *each* of its tokens (keywords). The answer to a publish request is the load of the peer addressed. We always answer with a very low load, thus we attract more and more publish requests to our sybils.

An 8-bit zone contains the peers whose KAD ID agrees in the first 8 bits, thus each zone can theoretically contain 2^{120} hash values. We actually observe between 12,000 and 25,000 peers per zone. The entire KAD network contains 256 8-bit zones and between 3 and 5 million peers. It is possible to spy on one zone of the KAD network only by restricting the returned KAD IDs to a certain prefix. Into a zone we can insert 65,356 distinct sybils to be sure to catch at least one of the ten publish messages for a keyword or a source and at least one of the three search messages that are sent per user-initiated search.

4 Performance Analysis

Let us first quantify the resources required to introduce sybil peers in the entire KAD ID space using *Mistral*. Three millions online peers that have to be crawled regularly with *Blizzard*, at least every two hours, to cope with the churn in the system. Each crawl accounts for 4 GByte of traffic. Afterward, the sybils must be announced to those peers. Assume that we only announce to each peer the 256 closest sybils: one announcement costs 50 bytes plus another 50 bytes for the ack; that accounts for $3,000,000 * 256 * 2 * 50$ bytes = 72 GByte. Announcements must be done periodically. On average, sybils announcement generates about 40 MBytes/s of traffic. Moreover, the sybils will also attract search and publish messages. That clearly shows that from the bandwidth point of view it is not possible, with our resources, to place sybils in the entire network.

Since spying on the entire KAD ID space was not possible, we spied on 20 different 8-bit zones of the KAD ID space during 24 hours. During this time, on average, 4.3 million publish messages, 350,000 search messages and 8.7 million route messages were recorded. The publish messages contained 26,500 different keywords per zone, most of them in latin letters, and 315,000 distinct sources, i.e., 315,000 distinct files. Among the 65,356 sybils we introduced, on the average 62,000 were hit by search or publish requests.

The hash values of the search requests, keywords publish requests and the keywords themselves are uniformly distributed over the KAD ID space. The same is true for the source search requests and source publish requests. Also we know from our earlier measurements with *Blizzard* that the peers are uniformly distributed as well on the KAD ID space [19].

This property allows us to estimate the total number S of sources (files) in the system by simply counting the number of sources in a zone. Let S_{part} be the number of sources counted in an 8–bit zone, and $\hat{S} := 256 * S_{part}$ the estimate for the total number of sources in the KAD system. Use Chernoff bounds (see [11] Chapter 4) we tightly bound the estimation error. Indeed, $Prob(|S - \hat{S}| < 45000) \geq 0.99$, which means that our estimate $\hat{S}$ has most likely an error of less than 3% for a total number of at least 80 million sources.

Our measurements show that there are ten times more publish messages than search messages. Moreover a publish message is 10 times bigger than a search message since it contains not only a keyword but also metadata describing the published content. Thus we focus on improving the performance of KAD by reducing the number of publish actions.

The number of times a keyword publication is observed versus the ranking of the keyword for the 8-bit zones `0xe3` and `0x8e` are shown in Figure 2 in log-log scale. Rank 1 is the most popular keyword. If each curve were to be a straight line, then the popularity of keywords would follow a Zipf-like distribution (i.e. the probability of seeing a publication message for the i'th most popular keyword is proportional to $1/i^{\alpha}$ [18]). We used Matlab's curve-fitting tools to estimate the value of α, for the curve. The value of α is the same for all zones: $\alpha \approx -1.63$. It is near the most popular keywords where the zones differ by an order of magnitude.

We picked two zones as examples. The zone `0xe3` contains the keyword "the" whereas the zone `0x8e` does not contain any popular keywords. The keyword "the" in zone `0xe3` accounts for 30% of the total load in the zone. In total 1,518,717 publish requests with the keyword "the" hit our sybils in 24 hours. Whereas, in zone `0x8e`, the most popular keyword accounts only for 5% of the load. In this zone the most popular keywords are nearly equally popular.

Not only the zone as a whole benefits from not publishing stopwords, but especially single peers close to the stopwords' hashes. Qiao and Bustamante [14] nicely describe the load balancing mechanism of Overnet, which relies on the same mechanisms as KAD. They state that peers close to hot spots are experiencing only 50% more overhead than other peers. However, all their measurements were done with search requests only, they only show how widely a keyword is spread over the hash space. Our measurements with *Mistral* however show that the publish messages always hit the machines close to the hash of the popular keyword first. Only if these machines are overloaded the publisher tries to contact machines more distant ones from the hash of the keyword. Figure 3 shows

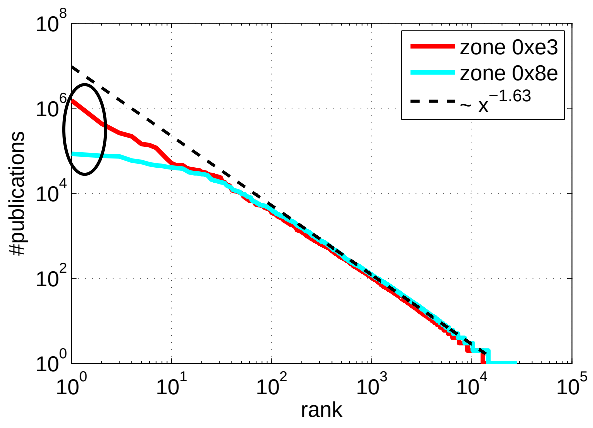

Fig. 2. The number of publications per keyword for two different zones.

the number of queries that hit our ten most loaded sybils in the two zones `0xe3` and `0x8e`. The popular keyword "the" in zone `0xe3` is mainly responsible for the high load on these syblis. The sybils with a lower rank have the same load in both zones.

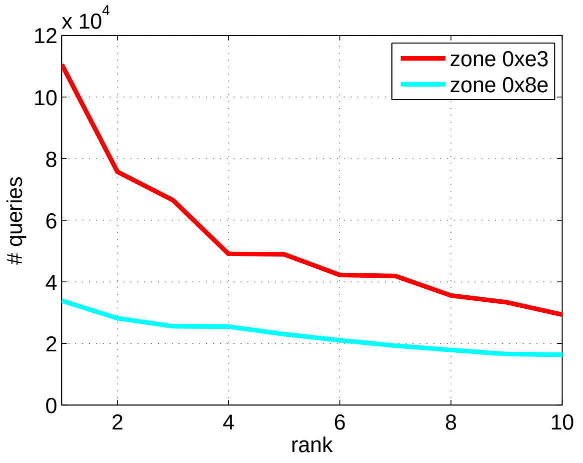

Fig. 3. The number of queries received by the sybils for two different zones.

The popular keywords that make the difference between the zones are all stopwords. Table 1(a) shows specific stopwords for KAD file names which augment the set believed to be used in Google (Table 1(b)). We propose to use the union of Table 1(a) and Table 1(b) as stopwords. The number of peers on which a stopword is published as well as the number of files containing the stopword have been determined by first crawling the peers around the stopword with *Blizzard* and then by querying all those peers for the stopword. Excluding these words from the publishing process does not affect the usability of the system, as a user can still specify the filetype (documents, music, movies, etc.) he is looking for.

(a) The stopwords for KAD

stopword	# peers	# files
avi	491	8101
xvid	479	13683
192kbps	437	8005
dvdscreener	413	12343
screener	433	7377
jpg	456	10529
pro	303	8378
mp3	482	12019
ac3	424	8045
video	468	10478
music	335	8558
rmvb	454	13643
dvd	450	10194
dvdrip	560	13235
english	388	7849
french	377	9468
dreirad	28	30

(b) The Google stopwords

stopword	# peers	# files
about	513	7608
are	330	7282
com	463	11550
for	549	12303
from	399	8345
how	542	8282
that	423	9148
the	487	14502
this	452	8510
what	394	7710
when	294	7241
where	431	9445
who	302	7742
will	458	7976
with	338	8543
www	391	11203
and	577	13706

Table 1. The Google and KAD stopwords with more than two letters, the number of peers storing them and the number of files containing them. For comparison the rare keyword “dreirad” is shown.

Unlike for the keywords, the popularity of the sources is much better balanced inside the zones and between the zones. The most popular source accounts only for 0.1% of the source publish traffic.

5 Related Work

Stopwords have been used for decades in indexing and retrieval; we thus concentrate on the use of stopwords in the context of peer-to-peer systems.

In [13] and [9] the authors describe an indexing system for the World Wide Web based on a peer-to-peer system. Their idea is that peer-to-peer systems should be able to build good search indexes in a distributed fashion, enabling Web searches in a much more scalable way than traditional (centrally coordinated distributed) search engines. These papers introduces the concept of

"Highly Discriminative Keys" (HDKs): these are a selection of "rare" keys occurring in a text document. Based on these HDKs, they build a peer-to-peer index which they then evaluate experimentally. They use up to 120,000 Reuters news articles; and then apply 250 English stopwords and a stemmer when they construct a search term from the full text. This is different from our approach: we do not intend to support full-text searches over the entire document, we are just trying to enhance the indexing for file names in peer-to-peer systems.

The authors of [2] present a general four-layer architecture for a peer-to-peer-based information retrieval system that is used to build a scalable index. They illustrate the functionality of their approach by describing how a concrete indexing system could be built with those four layers; in this context, stopwords are used to improve the precision on layer 4. Again, this architecture is more general, and aims at building a full-text search engine. Their system is not implemented, and no performance measurements are presented in the paper.

In [22], another approach for scalable Web search is proposed. The authors use a technique developed by R. Fagin to merge the result sets of single-term queries more efficiently than by just loading all of them onto a single site. Fagin's idea is based on sorted inverted lists that are efficiently merged across sites. The authors exclude stopwords in their searches, without specifying details. Experimental results with 120 million Web pages show a low communication overhead for multi-site queries based on Fagin's idea. However, it is not quite clear where peer-to-peer technology is used, and again, unlike in our system, the purpose is the construction of an efficient full-text index.

Detailed measurement results from a study of Gnutella and Overnet (a precursor of KAD) are presented in the paper of Qiao and Bustamante [14]. Among other things, the authors evaluate the performance of queries in Overnet. Of particular interest to our work are their results on queries to popular keywords (stopwords are very popular keywords). They conclude that these are handled well by Overnet because it distributes the query load to multiple peers whose hash IDs are "close enough" to the hash of the keyword: the more popular the keyword, the broader the hash range. Our measurements contradicted this conclusion: first, considerable overhead is generated by initially querying the peer with the closest hash to the popular keyword who answers with "too busy"; this goes on with an iteratively less precise hash value until a peer is found who is able to answer. Thus, a considerable additional load is imposed on the peers next to popular keywords. Second, we not only consider the querying but also the publishing load, which is much higher.

6 Conclusion

We have reported our findings obtained from spying on KAD, the largest currently deployed DHT. We developed *Mistral*, a content spy, that allows us to gain an overview of the content published and searched in KAD. Our observations show that the publication process in KAD is responsible for more than 90% of the total network traffic. Moreover we note that the load is highly unbalanced between the peers. The peaks of load are due to very popular keywords that are most often meaningless stopwords. We have then proposed to add a stopword exclusion step into all KAD based peer-to-peer systems. Our results show how this will equalize the load on the peers storing the keywords, and, as a consequence, improve the overall system performance. There is no draw back to stopword exclusion since stopwords do not carry much meaning.

References

1. A-Mule, http://www.amule.org/.
2. K. Aberer, F. Klemm, M. Rajman, and J. Wu, "An architecture for peer-to-peer information retrieval", In *Workshop on Peer-to-Peer Information Retrieval*, 2004.
3. A. T. Clements, D. R. K. Ports, and D. R. Karger, "Arpeggio: Metadata Searching and Content Sharing with Chord", In *International Workshop on Peer-To-Peer Systems*, 2005.
4. G. Danezis, C. Lesniewski-Laas, M. Kaashoek, and R. Anderson, "Sybil-resistant DHT routing", In *European Symposium On Research In Computer Security*, Springer, 2005.
5. J. Dinger and H. Hartenstein, "Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration", In *ARES '06: Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06)*, pp. 756–763, 2006.
6. J. R. Douceur, "The Sybil Attack", In *Proceedings of the 1^{st} International Workshop on Peer-to-Peer Systems (IPTPS)*, LNCS, pp. 251–260, March 2002.
7. E-Mule, http://www.emule-project.net/.
8. F. L. Fessant, S. B. Handurukande, A.-M. Kermarrec, and L. Massoulié, "Clustering in Peer-to-Peer File Sharing Workloads", In *IPTPS '04: The 3rd International Workshop on Peer-to-Peer Systems*, October 2004.
9. F. Klemm, A. Datta, and K. Aberer, "A Query-Adaptive Partial Distributed Hash Table for Peer-to-Peer Systems", In *Current Trends in Database Technology - EDBT 2004 Workshops*, pp. 506–515, 2004.
10. P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-peer Informatiion System Based on the XOR Metric", In *Proceedings of the 1^{st} International Workshop on Peer-to-Peer Systems (IPTPS)*, pp. 53–65, March 2002.
11. M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, Cambridge Press, 2005.
12. Overnet, http://www.overnet.org/.
13. I. Podnar, M. Rajman, T. Luu, F. Klemm, and K. Aberer, "Beyond Term Indexing: A P2P Framework forWeb Information Retrieval", *Informatica*, 30:153–161, 2006.
14. Y. Qiao and F. E. Bustamante, "Structured and Unstructured Overlays Under the Microscope - A Measurement-based View of Two P2P Systems That People Use.", In *Proceedings of the 2006 USENIX Annual Technical Conference*, 2006.
15. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network", In *Proc. ACM SIGCOMM*, 2001.
16. A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems", In *Proceedings of Middleware*, Heidelberg, Germany, November 2001.
17. R. Siebes, "pNear: combining Content Clustering and Distributed Hash Tables", *In Proceedings of the IEEE - p2pkm.org*, 2005.

18. K. Sripanidkulchai, “The popularity of Gnutella queries and its implications on scalability”, In *Proceedings of O’Reilly’s OpenP2P*, 2001.
19. M. Steiner, E. W. Biersack, and T. En-Najjary, “Actively Monitoring Peers in Kad”, In *Proceedings of the* 6th *International Workshop on Peer-to-Peer Systems (IPTPS’07)*, 2007.
20. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, “Chord: A scalable Peer-to-peer lookup service for Internet applications”, In *Proceedings of SIGCOMM*, pp. 149–160, San Diego, CA, USA, 2001, ACM Press.
21. D. Stutzbach and R. Rejaie, “Improving Lookup Performance over a Widely-Deployed DHT”, In *Proc. Infocom 06*, April 2006.
22. T. Suel, C. Mathur, J.-W. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram, “ODISSEA: A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval”, TR-CIS-2003-01, Department of Computer and Information Science, Polytechnic University, Brooklyn, NY, June 2003.