A flexible and robust lookup algorithm for P2P systems

Mauro Andreolini, **Riccardo Lancellotti**
University of Modena and Reggio Emilia
Modena, I-41100
{mauro.andreolini, riccardo.lancellotti}@unimore.it

Abstract

One of the most critical operations performed in a P2P system is the lookup of a resource. The main issues to be addressed by lookup algorithms are: (1) support for flexible search criteria (e.g., wildcard or multi-keyword searches), (2) effectiveness – i.e., ability to identify all the resources that match the search criteria, (3) efficiency – i.e. low overhead, (4) robustness with respect to node failures and churning. Flood-based P2P networks provide flexible lookup facilities and robust performance at the expense of high overhead, while other systems (e.g. DHT) provide a very efficient lookup mechanism, but lacks flexibility.

In this paper, we propose a novel resource lookup algorithm, namely fuzzy-DHT*, that solves this trade-off by introducing a flexible and robust lookup criteria based on multiple keywords on top of a distributed hash table algorithm. We demonstrate that the fuzzy-DHT algorithm satisfies all the requirements of P2P lookup systems combining the flexibility of flood-based mechanisms while preserving high efficiency, effectiveness ad robustness.*

Keywords: Peer-to-peer, Overlay Routing, Distributed Hash Tables, Resource lookup

1. Introduction

Peer-to-peer systems have emerged as a popular approach for the deployment of novel Internet-based services. According to literature [27, 15], a significant fraction of the Internet traffic is related to peer-to-peer applications. The peer-to-peer approach has been proposed for a wide set of applications, mainly due to their potential fault tolerance and scalability. Besides the popular file sharing networks, examples of peer-to-peer based services include multimedia streaming applications [32] or file systems [16, 10] designed for the management of massive amounts of data, Web caching [13], publishing systems [7], middleware architectures [25] and even distributed computing applications [28, 12]. The deployment of these services based on a peer-to-peer approach requires a lookup mechanism that is characterized by the following properties: *flexibility*, *effectiveness*, *efficiency*, and *robustness*. A flexible resource lookup algorithm allows complex queries, including keyword- and attribute-based searches. An effective resource lookup returns a list of all the resources that match the search predicate. An efficient resource lookup has the minimum impact on the underlying physical network. A robust mechanism can provide effective lookup even in the case when some nodes of the peer-to-peer network are faulty.

The most popular solutions for lookup algorithms are based on *centralized* or *flood-based* systems [20, 1] that provide an extremely flexible query semantics, such as wildcard-based searches. However, centralized solutions cannot provide robust lookup, while flood-based systems are characterized by a high traffic overhead and, consequently, a reduced efficiency. On the other hand, very efficient routing algorithms based on *distributed hash tables* (DHTs) [30, 22, 25] provide highly effective lookup, but provide poor query flexibility, because the lookup requires a unique identifier. Proposals for implementing keyword-based lookup on top of DHT have been proposed [23, 9], but these solution require the implementation of specific inverted index tables that are difficult to distribute and may reduce algorithm robustness.

In this paper, we propose a novel algorithm (*fuzzy-DHT*) that introduces a keyword-based search on top of a distributed hash table. This algorithm satisfies all the requirements of flexibility, effectiveness, efficiency and robustness: it supports a multiple keyword-based lookup semantics and it returns every resource in the overlay network matching the search criteria. The result is achieved with a contained overhead, because only the nodes that can potentially lead to a query hit are contacted during the lookup process. Furthermore, the algorithm can exploit the basic DHT solutions for fault tolerance, thus inheriting the robustness of these solutions. Our implementation of fuzzy-DHT is based on the

Plaxton routing algorithm of the Pastry [25] overlay network. However, we believe that the proposed algorithm can be adopted to extend the functions of services that already exploit the Pastry overlay such as Past or Scribe [7, 26].

To the best of our knowledge, this is the first algorithm that achieves the goal of combining flexibility, effectiveness, efficiency, and robustness. We propose a novel algorithm that enriches DHTs with flexible search criteria, while preserving their characteristics of effective lookup, low overhead and robustness with respect to failures. As a further advantage of our proposal, the proposed algorithm is easy to implement on top of existing DHT applications and does not require external services and indexes (such as external databases searches for hash keys or inverted indexes structures for routing).

We developed a simulator based on *ns-2* to compare the performance of the fuzzy-DHT algorithm with a standard flood-based search mechanism [1] and probabilistic flood algorithms [33, 21]. We show through extensive simulations that the proposed algorithm can provide a query effectiveness and robustness comparable with that of other search algorithm, with a considerably reduced communication overhead (at least an order of magnitude lower).

The rest of this paper is organized as follows. Section 2 discusses the related work on overlay routing algorithms. Section 3 introduces the fuzzy-DHT algorithm. Section 4 describes the testbed environment and the workloads used for the simulations. Section 5 presents a detailed comparison between the algorithms. Finally, Section 6 provides some concluding remarks.

2. Related work

The need for sophisticated resource lookup queries in peer-to-peer systems has first emerged in the file sharing networks [20, 1], where keyword-based searches are used to identify the files to download. The lookup process is typically based on flood-based algorithms [1] or on centralized directories [20]. However, the scalability limits of flood-based searches are well known, as shown in [8]. Scalability of flood-based lookup has been improved by means of semi-structured networks such as Kazaa [29], where flood-based lookup is carried out only within a small set of well-connected *supernodes*, but all these techniques cannot match the scalability characteristics achieved by means of more advanced approaches, such as distributed hash tables (DHTs), including the fuzzy-DHT algorithm proposed in this paper.

First generation DHTs [19, 22, 30, 25] are characterized by a simple operation semantics that allows to insert and retrieve key-value pairs. While providing extremely low overhead and guaranteeing a high scalability, this simple semantics cannot be applied to complex environment where sophisticated queries must be provided.

The fuzzy-DHT algorithm proposed in this paper addresses the issue of joining the need for advanced lookup features with the need to preserve the scalability of DHTs. Other studies propose flexible queries. For example, Liu *et al.* propose a system to support range queries [18], other researchers propose single keyword queries based on lookup over inverted indexes stored in the overlay network nodes [23, 14] and at least one example of multiple queries (with or composition) has been proposed in literature [9]. On the other hand, Tang *et al.* introduce semantic searches on the CAN DHT [31]. However, all these proposals require separate search services or introduce a completely new routing mechanism. Our approach is different from these proposals for three main reasons. First, the fuzzy-DHT algorithm allows the deployment of novel services with only slight modifications to the existing overlay networks, thus allowing a simpler deployment of the fuzzy-DHT-based overlay. Second, the proposed algorithm is explicitly designed to provide and-based multiple keyword searches, which are convenient for locating resources based on attributes. Finally, our algorithm is explicitly designed with efficiency and dependability as a primary goals.

3. The fuzzy-DHT algorithm

The fuzzy-DHT algorithm exploits the standard Plaxton-based lookup used in Pastry [25]. The support for flexible queries is introduced by adding two main features to the original algorithm: a new hash function that represents a list of keywords, and a query routing algorithm that *forks* the lookup process across nodes to ensure the identification of every suitable resource. Lookup effectiveness, efficiency and robustness are inherited from the standard Pastry routing algorithm.

3.1. Keyword representation

The proposed fuzzy-DHT algorithm introduces a first fundamental difference with respect to the original DHT algorithms in the hash function used to compute the resource keys. A hash function for the fuzzy-DHT algorithm must satisfy the following requirements:

- fixed-length binary representation of n;
- ability to represent the keywords $kw_1, kw_2, \ldots, kw_k$ associated with a resource;

We choose a Bloom Filter [6] as the data structure used for the key. The hash function for the fuzzy-DHT algorithm computes the Bloom filter. Let $B[]$ be an array of

n bits representing the key for the resource r (we assume the array $B[]$ is initially filled with 0). Let $KW[] = \{kw_1, kw_2, \ldots, kw_k\}$ be a set of keywords associated with resource r, and let $H[] = \{h_1, h_2, \ldots, h_m\}$ be a set of hash functions where $h_j : KW \rightarrow [0, n-1]$. The Bloom filter $B[]$ is built as follows: $B[h_j(kw_i)] \Leftarrow 1, \forall kw_i \in KW[], \forall h_j \in H[]$. For every keyword kw_i, we set to 1 the bits in the bloom filter corresponding to the results of the m hash functions computed on the keywords, as shown in Figure 1.

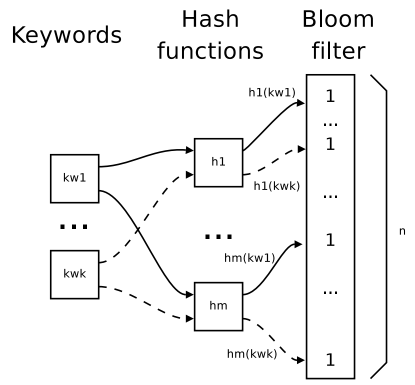


Figure 1. Bloom filter construction

The Bloom filter satisfies the two requirements of the hash function for the fuzzy-DHT algorithm. The representation of a Bloom filter is an array of fixed length, with n defined in the design phase of the overlay network. Since a Bloom filter is a compact representation of a data set, it can be used to store the association between a set of keywords $KW[]$ and the the corresponding resource identifier $B[]$. If all the bits associated with keyword kw are set, the keyword is likely to be associated with a resource. The above-mentioned condition can be formalized as: a necessary condition for the keyword kw to be represented in the Bloom filter $B[]$ is that $\sum_{\forall h_j \in H[]} B[h_j(kw)] = m$. This condition is necessary but not sufficient due to possible *Bloom filter collisions* that may cause false positives (also called false hits). However, the literature proposing solutions based on Bloom filters suggests that the percentage of false positives due to collisions is usually low if proper filter tuning is applied [11, 24].

3.2. Overlay routing algorithm

Before describing the fuzzy-DHT algorithm, we briefly recall the main features of the original Pastry algorithm. In the Pastry system, nodes and resources are identified by means of a key. The key is a fixed-length string of bits that acts as a unique identifier for nodes and resources. Most DHTs, including Pastry, suggest to use digests to generate a key. For example, the key of a node can be computed as the hash (MD5 or SHA1) of its IP address; the key of a resource may be computed as the hash of the resource identifier (e.g., in a file-sharing context, the filename).

In Pastry a key is represented as a sequence of d digits, each composed by b bits. We represent a key $KX = (x_1, x_2, \ldots, x_d)$ as a vector with d elements (each corresponding to a digit). Resources and nodes share the same hash space and are identified by a key value. Each resource is hosted by the node with the key closer to the resource key value. Lookup requests are routed by the overlay network in order to reach the node that hosts the resource itself. It is important to note that in the Pastry algorithm, as in most DHTs, the query refers to a key and only the resource with the exact key is reached. The lookup uses a longest prefix matching algorithm that tries at every step to increase by 1 digit the length of the matching prefix.

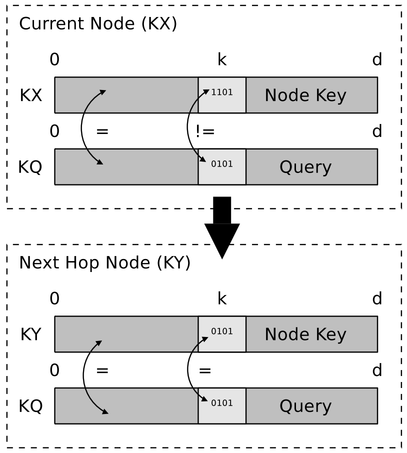


Figure 2. The Pastry lookup process

Figure 2 shows the routing step of the Plaxton algorithm used in the Pastry overlay network. Let $KQ = (q_1, q_2, \ldots, q_k, \ldots, q_d)$ be the key in a lookup request reaching node X with key $KX = (x_1, x_2, \ldots, x_k, \ldots, x_d)$. We suppose that the matching prefix has a length of $k-1$. This implies that $q_i = x_i, \forall i \in [1, k-1]$ and $q_k \neq x_k$ (see the left side of Figure 2).

The lookup step uses the digit x_k of the key and a set data structure called neighbor and leaf tables to identify the next hop Y. This node has a key $KY = (y_1, y_2, \ldots, y_k, \ldots, y_d)$ such that $q_i = y_i \forall i \in [1, k]$, as shown in the right part of Figure 2. We omit further details of the Pastry algorithm for the sake of simplicity. The reader can refer to [25] for a complete descrition.

The second significant contribution of the proposed overlay routing algorithm is the process used to select the next

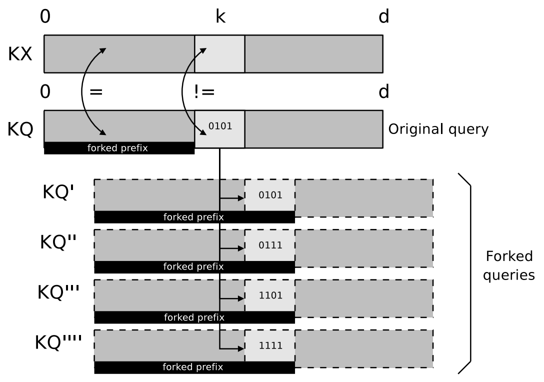


Figure 3. Query forking process

hop in the overlay routing process. The fuzzy-DHT algorithm extends the Plaxton-based routing algorithm to take advantage of the enhanced hash function used in the resource key computation.

The fuzzy-DHT algorithm allows a multiple keyword-based lookup such that the query is forwarded to every node that hosts resources whom key contains *all* the keywords composing the search key. The resources that result in a hit are characterized by a *superset* of the keywords composing the query. Recalling the definition of the Bloom filter, matching all the keywords means that each bit with a value of "0" in the query key KQ acts as a *wildcard* for matching purposes. We introduce a new operation in the query routing step, namely *query fork*, for the management of the wildcard bit. During the query fork we generate a set of new queries that have every possible combination of bit in place of the wildcard bits in the original query. A complete query fork (where every "0" in the key is used for forking) at the beginning of the route process would saturate the network. We choose to fork queries only when it is required, in order to save network resources. At every routing step, only the key digit x_k is subject to fork. Figure 3 provides an example of the query fork process: the node with key KY receives the query KQ; the first $k-1$ in both KQ and KY are identical. For the k-th digit with value 0101 a query fork is necessary. Considering the 0 in the k-th digit we obtain four values (0101, 0111, 1101, 1111) that are used to generate four *forked queries* KQ', KQ'', KQ''', KQ''''. The forked queries are then handled as standard queries and are routed according to the Plaxton-based algorithm.

Algorithm 1 shows the fuzzy-DHT routing algorithm. Even if the proposed algorithm introduces a significant change in the functionality of the original Pastry algorithm, the implementation effort remains limited to a few lines of code related to the management of the query fork process. The *pastry_next_hop()* function (the most complex part of the routing algorithm) is identical to the Plaxton-based algorithm used in Pastry. The simplicity of fuzzy-DHT is one of its distinctive features because it allows to take advantage of pre-existing code base. Indeed, in our experiments, the simulator code for the fuzzy-DHT algorithm reuses nearly 90% of the code of the Pastry algorithm. Furthermore, the fuzzy-DHT algorithm reuses all the mechanisms for self-configuration and fault-tolerance already available in Pastry, thus inheriting the proven robustness of this algorithm.

Algorithm 1 shows also the details of the query management process in the fuzzy-DHT algorithm. The prefix length k is calculated in the loop in lines 1-4 of the algorithm. The *if* statement in line 5 is required for a proper handling of the query fork even in the case where the next hop must reconsider the same x_k digit of the previous routing step. Indeed, in the case where neighbor table is incomplete, the Pastry algorithm can forward a query to a nearby node without increasing the matching prefix length [25]. The fuzzy-DHT algorithm must avoid unneeded query forking even in this case. To this purpose, the fuzzy-DHT algorithm introduces an attribute to the key, that we call *forked prefix*. The forked prefix represents a pointer to the last digit in the key where query forking occurred. The query forking process occurs for byte k only if the forked prefix states that no forking already occurred for that byte. The forked prefix starts with a value of 0 and increases by one digit for every key fork, as shown in line 7 of Algorithm 1. The query fork process (shown in Figure 3) is used to create a new set of query keys (the forked queries), to which we associate the new value of the $forkedPrefix$ variable. The forked queries are then routed to their next hop according to the standard Plaxton-based algorithm. When the routing must consider for two or more consecutive hops the same digit x_k, the query fork occurs at most once, because in the next hops the forked prefix will have a length of k and will prevent additional query forks. Hence, the query key is routed according to the standard Pastry algorithm.

Algorithm 1 fuzzy-DHT algorithm

```
Require: $forkedPrefix, KQ = \{q_0, q_1, \ldots, q_n\}, KX = \{x_0, x_1, \ldots, x_n\}$
Ensure: Step of overlay routing (fuzzy-DHT)
1: $k \Leftarrow 0$
2: while $q_k = x_k$ do
3:     $k \Leftarrow k + 1$
4: end while
5: if $forkedPrefix < k$ then
6:     $ForkedKQ \Leftarrow$fork_query($KQ, k$)
7:     $forkedPrefix \Leftarrow forkedPrefix + 1$
8:     for all $KQ' \in ForkedKQ$ do
9:         pastry_next_hop($KQ'$)
10:    end for
11: else
12:    pastry_next_hop($KQ$)
13: end if
```

4. Experimental testbed

The description of the *fuzzy-DHT* algorithm in Section 3 demonstrates that the algorithm guarantees search flexibility thanks to the support for multiple keywords in the lookup process. We now evaluate the effectiveness, the efficiency and the robustness of the fuzzy-DHT algorithm through extensive simulations. To this purpose, we extended the popular *ns-2* simulator with the support for the fuzzy-DHT, the flood-based and the probabilistic-flood algorithms. The flood-based algorithm is similar to the algorithm used in the popular Gnutella file sharing network [1], while the probabilistic flood algorithm is a variation of the standard flood-based algorithm where at each step and for each neighbor, the propagation of the query occurs with a given probability π [33, 21] (the standard flood-based algorithm is an extreme case of probabilistic routing with $\pi = 1$). In our experiments we use $\pi = 0.7$, because preliminary experiments demonstrate that this value ensures an high effectiveness, guaranteeing an higher efficiency with respect to the standard flood-based algorithm. For both flood-based and probabilistic flood algorithms, we consider an overlay network where neighborhood relationship creates a scale-free network following the Barabasi and Albert model [4, 5]. Indeed, scale-free networks are characterized by a power-law distribution of neighboring degree that is typical of real peer-to-peer scenarios [2].

The simulator focuses on query routing and does not take into account other operations of peer-to-peer networks, such as the fruition of the looked-up resource, e.g., the download in the case of file sharing networks. The simulator takes care of the setup of the physical network, the generation of initial conditions for the overlay network, and the generation queries issued by the peers of the network. We also implement the logic for the management of the overlay network management that takes into account node joins and leaves, as well as the support for managing the network when a faulty node is detected. These operation are directly provided by the Pastry protocol, as discussed in [25].

We use UDP as the transport layer for every considered overlay network because the query-response mechanism proper of overlay network operations is easily mapped on UDP messages; moreover most new-generation P2P systems are progressively discarding TCP-based transport in favor of the more agile UDP [1, 19]. Each experiment is carried out 10 times (each time using traces obtained from different random seeds) and the results are averaged over the runs to guarantee that the results are statistically relevant. The resources are randomly distributed over the nodes of the overlay network according to a normal distribution with mean value of 10 resource per node. Each resource is described by 5 words on average, according to the workload characterization available in literature [3]. In order to ensure the correct behavior of the DHT, we consider that after the initialization of the network, the DHT can re-distribute the resources over the nodes according to the routing algorithm.

Each simulation run generates 10^6 queries that are issued by the nodes of the simulation network. For each query we randomly select a number of keywords based on the workload characterization in [3]. From each query we compute the bloom filter representing the keywords and we compute the number of bits set to "1" in the filter. We define the ratio between the number of bits set to "1" and the bloom filter length as the query selectivity σ. The query selectivity represents the number of potential hits returned by a query. When σ is low, the query is highly selective and only few resources will match with the query predicate. On the other hand, when σ is high, most resource will generate a hit.

5. Performance evaluation

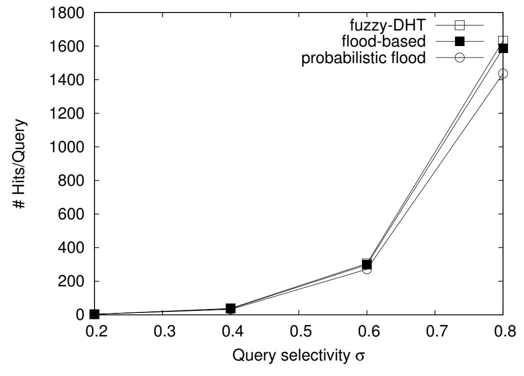
In this section we show the performance of the fuzzy-DHT algorithm by comparing its effectiveness and efficiency with flood-based search algorithms. We demonstrate that the fuzzy-DHT algorithm provides the best effectiveness and efficiency for every considered scenario of query selectivity σ, network size and percentage of node failures.

5.1. Impact of the query selectivity

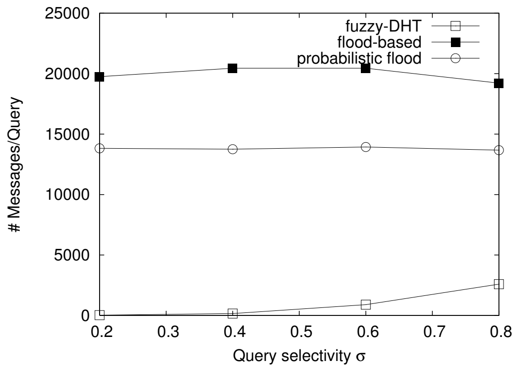
We now compare the three algorithms by evaluating the effectiveness and the efficiency as a function of the query selectivity parameter σ.

Fig. 4 compares the performance of the fuzzy-DHT, flood-based, and probabilistic flood algorithms as σ ranges from 0.2 to 0.8 over a network of 500 nodes. The σ is a function of the number of keywords used in the search. The typical cases of resource lookup in file sharing networks and in Web caching is using 3 or 4 keywords [23, 17], which results in a value of σ close to 0.7. We recall that the higher is σ the lower is the query selectivity, hence a query with $\sigma = 0.7$ is likely to provide a significant number of hits.

Fig. 4(a) compares the effectiveness of the three protocols. As σ grows, the number of hits for each query increases. The graph shows that fuzzy-DHT and flood-based algorithms achieve a similar effectiveness, which is within 5% from the optimal theoretical value (obtained by comparing the query traces with the resources available on the network), while the probabilistic flood-based algorithm is less effective, with a penalty of nearly 10% with respect to the optimal theoretical value. If we consider the overhead, shown in Fig. 4(b), the differences between the three algorithms are evident. The overhead of the flood-based query is up to three order of magnitude higher than that of the fuzzy-DHT algorithm. The probabilistic-flood algorithm achieves a higher efficiency if compared to the flood-based algo-



(a) Effectiveness



(b) Efficiency

Figure 4. Impact of the query selectivity σ

rithm, with an overhead nearly halved with respect to the flood-based algorithm, but still significantly higher if compared to the fuzzy-DHT algorithm. The overhead of the flood-based and probabilistic flood algorithms is similar for every value of σ. This can be explained by considering that the query propagation is completely unrelated to the matching process that leads to resource discovery. On the other hand, the fuzzy-DHT algorithm sends queries only to the nodes that may return a hit. As a consequence, the overhead of this algorithm grows as σ increases. However, even for high values of σ (e.g., $\sigma = 0.8$) the overhead of the fuzzy-DHT algorithm remains at least one order of magnitude lower than that of the flood-based lookup.

5.2. Scalability

Scalability analysis evaluates the impact of the overlay network size over the effectiveness and efficiency of the considered algorithms. We carry out the analysis using a network topology with a variable number of nodes issuing queries with $\sigma = 0.6$.

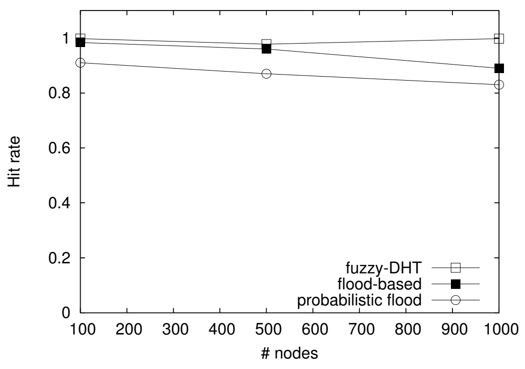
Fig. 5 shows the effectiveness and the efficiency of the fuzzy-DHT, flood-based and probabilistic flood algorithms. Fig. 5(a) shows the hit rate as the measure of the lookup effectiveness instead of the number of hits per query used in the previous analysis. The motivation is that as nodes are added to the network, the number of resources available grows, hence we normalize the number of hits per query against the theoretical maximum hit rate. We confirm that every algorithm is effective, with hit rates always higher than 80%. However, the flood-based and the probabilistic flood algorithms present an hit rate degradation for large network sizes. This effect is motivated by the query time-to-live that does not guarantee the exploration of the whole network when the number of nodes is very high. On the other hand, the fuzzy-DHT lookup algorithm achieves the best scalability with no effectiveness degradation with respect to the network size.

Fig. 5(b) shows the overhead per query as a function of the network size. We observe that every algorithm shows an increment in the traffic generated with each query. However, the overhead growth of the flood-based and probabilistic flood algorithms is much more evident than the overhead growth of the fuzzy-DHT algorithm. Indeed, the overhead of the flood-based and probabilistic flood algorithms grows by a factor of 10 as the number of nodes increases from 100 to 1000 because, for every query, the number of nodes to visit is higher. On the other hand, the overhead increase for the fuzzy-DHT is only by a factor of 4. The better scalability is due to the ability of the algorithm to route queries only to a reduced fraction of nodes that have a high probability of hosting the requested resource.

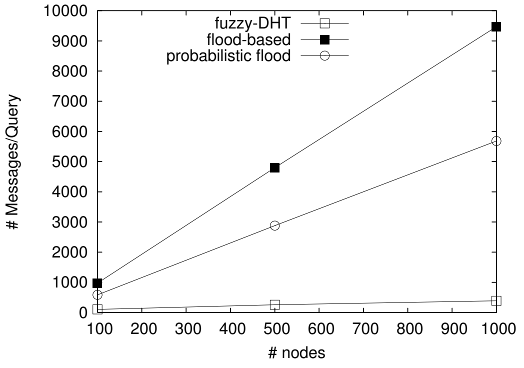
5.3. Robustness

Robustness evaluation is carried out by issuing queries on the overlay network ($\sigma = 0.6$) and introducing failures in the network nodes. A variable fraction of nodes is selected and marked as faulty in each experiment. A faulty node cannot report any hit and the hosted resources are considered lost. Furthermore, during the routing operation, a faulty node cannot act as the next hop for the routing process and must be discarded from neighbor tables.

Fig. 6(a) shows the number of hits for each query as a function of the amount of faulty nodes in the network. As expected, the number of hits is reduced as the fraction of faulty nodes grows for every algorithm. The main motivation for the hit rate reduction is the loss of resources hosted on the faulty nodes, because both the inherent message redundancy of flood-based protocols and the alternate routing strategies of Plaxton's algorithm are effective in preserving search effectiveness. On the other hand, the faulty nodes determine a growth in the amount of exchanged messages, although this impact is almost negligible if compared with the normal traffic generated by the algorithms. Fig. 5(b) shows the overhead per query as a function of the amount of faulty

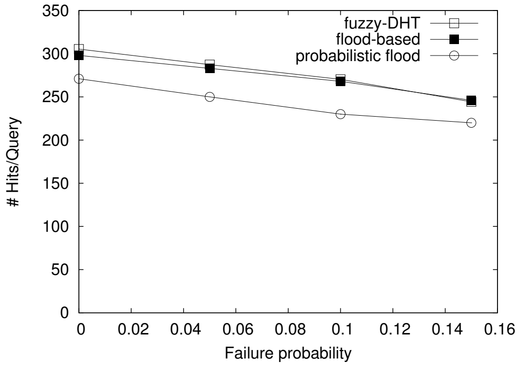


(a) Effectiveness

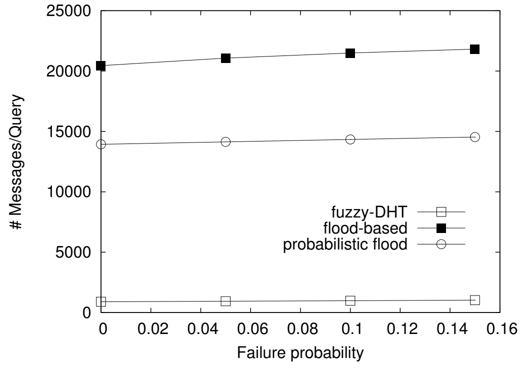


(b) Efficiency

Figure 5. Scalability evaluation



(a) Effectiveness



(b) Efficiency

Figure 6. Robustness evaluation

nodes and demonstrates that the amount of traffic generated by fuzzy-DHT algorithm remains nearly two order of magnitude lower with respect to the other alternatives even when 15% of the nodes are faulty.

6. Conclusions

The P2P approach has gained popularity in the context of distributed services, and multiple novel applications are being deployed using this scheme. The lookup of resources in a highly distributed environment remains one of the most critical tasks of peer-to-peer applications.

Current lookup algorithms must address a trade-off issue between flexibility, efficiency and robustness: the flexible keyword-based lookup of the flood-based algorithms is provided at the expenses of high overhead, while the efficient DHT-based lookup is not flexible or rely on data structures such as reverse indexes that may hinder algorithm robustness. In this paper, we have shown that it is possible to design an novel algorithm that introduces keyword-based lookup operations on top of an efficient and robust distributed hash table. The implementation does not require significant modification to the native source code of existing overlay network lookup algorithms. Our experiments show that the proposed fuzzy-DHT algorithm meets the main requirements for the most innovative P2P applications.

- The keyword-based lookup of the fuzzy-DHT algorithm provides the flexibility of innovative applications and services based on a peer-to-peer approach.
- The fuzzy-DHT algorithm is effective, as it achieves hit rates higher than flood-based search and very close to the theoretical optimum.
- It is extremely efficient as it limits the lookup overhead to at least one order of magnitude lower than that of flood-based and probabilistic flood algorithms. Our experiments proves that the high efficiency of the proposed algorithm is guaranteed for every considered query selectivity, and physical network topology. Furthermore, the efficiency of the fuzzy-DHT algorithm guarantees higher scalability with respect to the other considered protocols.
- The proposed algorithm is as robust as flood-based solutions thanks to the Pastry network management algorithms that address the issues of node failures, leaves and joins.

References

[1] Rfc-gnutella 0.6, 2008. http://rfc-gnutella.sourceforge.net/.

[2] M. Amoretti, F. Zanichelli, and G. Conte. Performance evaluation of advanced routing algorithms for unstructured peer-to-peer networks. In *Proc. of Valuetools 2006*, Pisa, Italy, Oct. 2006.

[3] M. Andreolini, R. Lancellotti, and P. S. Yu. Analysis of peer-to-peer systems: workload characterization and effects on traffic cacheability. In *Proc. of 12th Annual Meeting of the IEEE / ACM MASCOTS 2004*, Volendam, NL, Oct. 2004.

[4] A. Barabasi and R. Albert. Emengence of scaling in random networks. *Science*, 286(5439):509–512, Oct. 1999.

[5] A. Barabasi, R. Albert, and H. Jeong. Mean-field theory for scale-free random networks. *Physica A*, 272(1–2):173–187, Oct. 1999.

[6] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, Jul. 1970.

[7] M. Castro, P. Druschel, and A. Rowston. Scalable application-level anycast for highly ynamic groups. In *Proc. of NGC 2003*, Munich, Germany, Sept 2003.

[8] Y. Chawathe, S. Ratnasamy, L. Breslau, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proc. of ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Karlsruhe, Germany, Aug. 2003.

[9] H. Chen, H. Jin, J. Wang, L. Chen, Y. Liu, and L. M. Ni. Efficient multi-keyword search over p2p web. In *Proc. of the 17th international conference on World Wide Web*, 2008.

[10] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, Oct. 2001.

[11] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. on Networking*, 8(3):281–293, Jun 2000.

[12] R. Gupta, V. Sekhri, and A. K. Somani. Compup2p: An architecture for internet computing using peer-to-peer networks. *IEEE Tran. on Parallel and Distributed Systems*, 17(11), Nov. 2006.

[13] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized, peer-to-peer web cache. In *Proc. of 21st ACM Symposium on Principles of Distributed Computing (PODC 2002)*, Monterey, CA, Jul. 2002.

[14] Y.-J. Joung, C.-T. Fang, and L.-W. Yang. Keyword search in dht-based peer-to-peer networks. In *Proc. of 25th IEEE Int'l conf. on Distributed Computing Systems (ICDCS'05)*, Columbus, OH, Jun. 2005.

[15] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. Is p2p dying or just hiding? In *Global Internet and Next Generation Networks*, 2004.

[16] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proc. of ACM ASPLOS*, Nov. 2000.

[17] N. Leibowitz, A. Bergman, R. Ben-Shaul, and A. Shavit. Are file swapping networks cacheable?: Characterizing p2p traffic. In *Proc. of 7th Int'l Workshop on Web Content Caching and Distribution (WCW '02)*, Boulder, CO, USA, Aug. 2002.

[18] B. Liu, W.-C. Lee, and D. L. Lee. Supporting complex multi-dimensional queries in p2p systems. In *Proc. of 25th IEEE Int'l conf. on Distributed Computing Systems (ICDCS'05)*, Columbus, OH, Jun. 2005.

[19] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. of the 1st Int'l Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA., Mar. 2002.

[20] Napster. Napster file sharing system, 2008. http://www.napster.com.

[21] M. Portmann and A. Seneviratne. The cost of application-level broadcast in a fully decentralized peer-to-peer network. In *Proc. of the 7th Int'l Symposium on Computers and Communications (ISCC'02)*, 2002.

[22] S. P. Ratnasamy. A scalable content-addressable network. In *In Proceedings of ACM SIGCOMM*, pages 161–172, 2001.

[23] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proc. of the ACM/IFIP/USENIX Middleware Conference*, Rio de Janeiro, Brazil, Jun. 2003.

[24] A. Rousskov and D. Wessels. Cache digests. *Computer Networks and ISDN Systems*, 30(22-23), Nov. 1998.

[25] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, nov 2001.

[26] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proc. of 18th ACM SOSP'01*, Lake Louise, Alberta, Canada, Oct. 2001.

[27] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, Jan. 2002.

[28] SETI. Seti@home, 2008. http://setiathome.berkeley.edu/.

[29] Sharman. Kazaa file sharing system, 2008. http://www.kazaa.com.

[30] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[31] C. Tang, Z. Xu, and M. Mahalingam. psearch: information retrieval in structured overlays. *SIGCOMM Comput. Commun. Rev.*, 33(1):89–94, 2003.

[32] D. A. Tran, K. A. Hua, and T. Do. Zigzag: an efficient peer-to-peer scheme for media streaming. In *Proc. of 22nd Annual Joint Conference on IEEE Computer and Communications (IEEE Infocom)*, S. Francisco, CA, Mar. 2003.

[33] M. B. Yassein, M. Ould-Khaoua, and S. Papanastasiou. On the performance of probabilistic flooding in mobile ad hoc networks. In *Proc. of the 11th Int'l Conference on Parallel and Distributed Systems (ICPADS'05)*, 2005.