Enabling Dynamic Querying over Distributed Hash Tables

Domenico Talia[a,b], Paolo Trunfio[*,a]

[a] *DEIS, University of Calabria, Via P. Bucci 41C, 87036 Rende (CS), Italy*
[b] *ICAR-CNR, Via P. Bucci 41C, 87036 Rende (CS), Italy*

Abstract

Dynamic querying (DQ) is a search technique used in unstructured peer-to-peer (P2P) networks to minimize the number of nodes that is necessary to visit to reach the desired number of results. In this paper we introduce the use of the DQ technique in structured P2P networks. In particular, we present a P2P search algorithm, named DQ-DHT (Dynamic Querying over a Distributed Hash Table), to perform DQ-like searches over DHT-based overlays. The aim of DQ-DHT is twofold: allowing arbitrary queries to be performed in structured P2P networks and providing dynamic adaptation of the search according to the popularity of the resources to be located. DQ-DHT has been particularly designed for use in those distributed environments, like computational grids, where it is necessary to support arbitrary queries for searching resources on the basis of complex criteria or semantic features. This paper describes the DQ-DHT algorithm using Chord as basic overlay and analyzes its performance in comparison with DQ in unstructured networks.

Key words: Dynamic Querying, Distributed Hash Tables, Structured Peer-to-Peer Networks

1. Introduction

Information services are fundamental components of distributed systems as they allow us to locate the resources needed to execute large-scale parallel

[*] Corresponding author
Email addresses: talia@deis.unical.it (Domenico Talia), trunfio@deis.unical.it (Paolo Trunfio)

and distributed applications based on application requirements and resource availability. Designing a decentralized but efficient information service is a significant strand of research, with many researches demonstrating the use of peer-to-peer (P2P) models and techniques as an effective alternative to centralized and hierarchical solutions [1]. Such P2P systems are typically classified as *structured* or *unstructured*, based on the way nodes are linked to each other and information about resources is placed across the resulting overlay.

Structured systems, like Chord [2], Pastry [3], and Tapestry [4], use a Distributed Hash Table (DHT) to assign to each node the responsibility for a specific part of the resources. When a peer wishes to locate a resource identified by a given key, the DHT allows us to locate the node responsible for that key in $O(\log N)$ hops using only $O(\log N)$ neighbors per node. In unstructured systems, like Gnutella [5] and Kazaa [6], links among nodes can be established arbitrarily and data placement is unrelated to the topology of the resulting overlay. To locate a given resource, the query must be distributed through the network to reach as many nodes as needed. Each node reached by the query processes it on the local resources and, in the case of match, replies to the query initiator.

Thanks to the DHT infrastructure, searching in structured systems is more efficient - in terms of traffic generated - than searching in unstructured systems. As compared to unstructured P2P systems, however, structured systems provide a limited support to complex queries. Although several extensions to basic DHT schemes have been proposed to support, for instance, range queries [7], multi-attribute search [8], and keyword-based search [9], DHT-based lookups still do not support arbitrary queries (e.g., regular expressions [10]) since it is infeasible to generate and store keys for every query expression. On the other hand, unstructured systems can do it effortlessly since all queries are processed locally on a node-by-node basis [11].

Even if the lookup mechanisms of DHT-based systems do not support arbitrary queries, it is possible to exploit their structure to distribute any kind of information across the overlay with minimal cost. For example, in [12] a technique to perform an efficient broadcast over a DHT is proposed. Using such a technique, a broadcast message originating at an arbitrary node in the DHT overlay reaches all other nodes without redundant messages in $O(\log N)$ steps. It can be used to broadcast arbitrary types of queries, which can then be processed locally by single nodes as in unstructured systems. We elaborate on such an approach by proposing a P2P search algorithm,

named DQ-DHT (Dynamic Querying over a Distributed Hash Table), to provide efficient execution of arbitrary queries in structured P2P networks. DQ-DHT is based on a combination of the broadcast technique mentioned above with the dynamic querying (DQ) technique [13] used in unstructured networks.

The goal of DQ in unstructured networks is to minimize the number of nodes that is necessary to visit in an unstructured network to obtain the desired number of results. Using the DQ technique, the query initiator starts the search by sending the query to a few of its neighbors and with a small time-to-live (TTL). The main goal of this "probe" query is to estimate the popularity of the resource to be located. If such an attempt does not produce a sufficient number of results, the search initiator sends the query towards the next neighbor with a new TTL. Such TTL is calculated taking into account both the desired number of results and the resource popularity estimated during the previous phase. This process is repeated until the expected number of results is received, or there are no more neighbors to query.

Similarly to DQ, DQ-DHT performs the broadcast in an iterative way until the target number of results is obtained. At each iteration, a new subset of nodes is queried on the basis of the estimated resource popularity and the desired number of results. In contrast to DQ, DQ-DHT exploits the structural properties of the DHT to avoid message duplications and ensure higher success rate.

DQ-DHT has been particularly designed to serve as resource discovery mechanism for decentralized infrastructures like computational grids. Large-scale grids are typically organized into multiple domains, often referred to as virtual organizations (VOs) [14]. Within each VO, one node is typically designated as information server to answer queries about all the resources belonging to that domain. Since information servers are highly reliable nodes, it is possible to build a P2P network of information servers having a significantly lower churn rate than typical P2P networks. Thus, we consider a scenario in which the DHT overlay is composed of information servers only, which ensures a high stability of the overlay even in large-scale networks.

In this paper we describe the DQ-DHT algorithm using Chord as the DHT overlay. We analyze the performance of DQ-DHT through simulations under different algorithm configurations. We also compare the performance of DQ-DHT with that of DQ in unstructured networks. The simulation results show that DQ-DHT generates much less network overhead (i.e., number

of messages) than DQ, with a comparable - and in some cases better - search time, and with a higher success rate when the resource to be found is rare. An early version of this paper appeared as [15]. This new version includes additional experiments on the precision of the formulas used in DQ-DHT, a more detailed discussion of the algorithm, an extended performance evaluation, and a related work section.

The rest of the paper is organized as follows. Section 2 provides a background on the technique of broadcast over a DHT exploited by DQ-DHT. Section 3 describes the DQ-DHT algorithm. Section 4 analyzes its performance and compares DQ-DHT with DQ. Section 5 discusses related work. Finally, Section 6 concludes the paper.

2. Background on broadcast over a Chord overlay

As outlined above, DQ-DHT combines the DQ technique with a technique for efficient broadcast over a DHT. In order to better explain how DQ-DHT works, this section provides a background on the Chord-based implementation of the broadcast algorithm designed by El-Ansary et al., as proposed in [12].

Chord uses a consistent hash function to assign to each node an m-bit identifier, which represents its position in a circular identifier space ranging from 0 to $2^m - 1$. Each node, x, maintains a *finger table* with m entries. The j^{th} entry in the finger table at node x contains the identity of the first node, s, that succeeds x by at least 2^{j-1} positions on the identifier circle, where $1 \leq j \leq m$. Node s is called the j^{th} *finger* of node x. If the identifier space is not fully populated (i.e., the number of nodes, N, is lower than 2^m), the finger table contains redundant fingers. In a network of N nodes, the number u of unique (i.e., distinct) fingers of a generic node x is likely to be $\log_2 N$ [2]. In the following, we will use the notation F_i to indicate the i^{th} *unique finger* of node x, where $1 \leq i \leq u$.

To broadcast data item D, a node x sends a BROADCAST message to all its unique fingers. The BROADCAST message contains D and a *limit* argument, which is used to restrict the forwarding space of a receiving node. The *limit* sent to F_i is set to F_{i+1}, for $1 \leq i \leq u-1$. The *limit* sent to the last unique finger, F_u, is set to the identifier of the sender, x. When a node y receives a BROADCAST message with a data item D and a given *limit*, it is responsible for forwarding D to all its unique fingers in the interval $]y$, *limit*$[$. When forwarding the message to F_i, for $1 \leq i \leq u-1$, y supplies it a new

limit, which is set to F_{i+1} if it does not exceed the old *limit*, or the old *limit* otherwise. As before, the new *limit* sent to F_u is set to y. As shown in [12], in a network of N nodes, a broadcast message originating at an arbitrary node reaches all other nodes in $O(\log_2 N)$ steps and with exactly $N-1$ messages.

Figure 1a shows an example of broadcast in a fully populated Chord ring, where $u = m = 4$. For each node, the corresponding finger table is shown. The BROADCAST messages are represented by rectangles containing the data item D and the *limit* parameter. The entire broadcast is completed in $u = 4$ steps, represented with solid, dashed, dashed-dotted, and dotted lines, respectively.

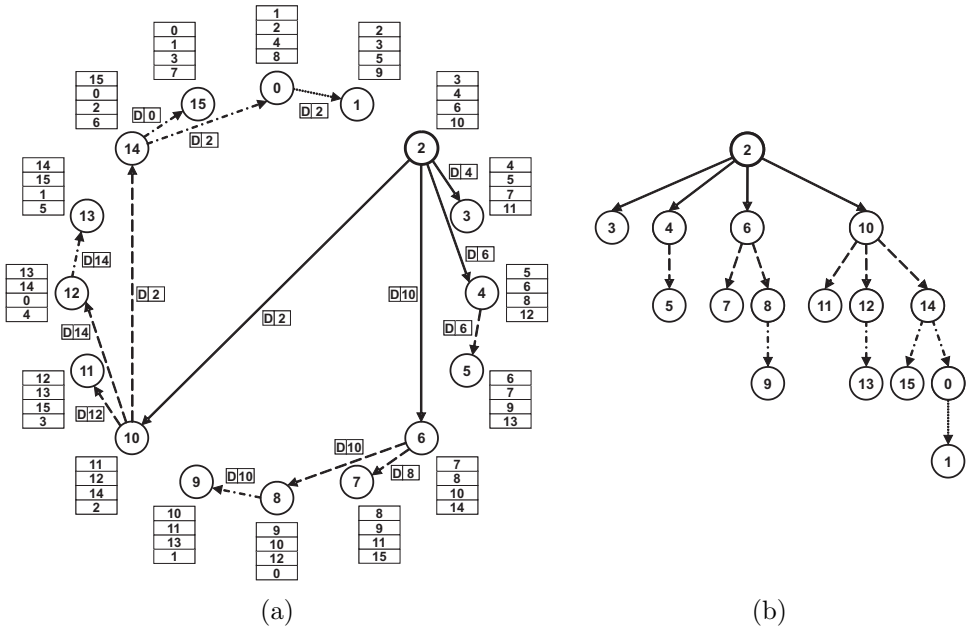


Figure 1: (a) Example of broadcast in a fully populated Chord ring; (b) corresponding spanning tree.

In this example, the broadcast is initiated by *Node 2*, which sends a BROADCAST message to all nodes in its finger table (*Nodes 3, 4, 6* and *10*) (*step 1*). *Nodes 3, 4, 6* and *10* in turn forward the BROADCAST message to their fingers under the received *limit* value (*step 2*). The same procedure applies iteratively, until all nodes in the network are reached (*steps 3* and *4*).

The overall broadcast procedure can be viewed as the process of passing the data item through a spanning tree that covers all nodes in the net-

work [12]. Figure 1b shows the spanning tree corresponding to the example of broadcast shown in Figure 1a. The root of the spanning tree is the node that initiates the broadcast (*Node 2*). The tree is composed of four subtrees, each one having, as root, one of the fingers of *Node 2* (that is, *Nodes 3, 4, 6* and *10*). Since the spanning tree corresponds to the lookup tree, which is a *binomial tree* in a (fully populated) Chord network [16], also the spanning tree associated with the broadcast over a fully populated Chord ring is a binomial tree.

3. Dynamic Querying over a DHT

As mentioned earlier, the goal of DQ-DHT is twofold: allowing arbitrary queries in a Chord DHT and supporting dynamic adaptation of the search based on the popularity of the resources to be located. To support dynamic search adaptation, DQ-DHT performs the search in an iterative way, similarly to the original DQ algorithm introduced in Section 1. In the following, the DQ-DHT algorithm is briefly described.

Let x be the node that initiates the search, U the set of unique fingers not yet visited, and R_d the desired number of results. Initially U includes all unique fingers of x. Like in DQ, the search starts with a probe query, aimed at evaluating the popularity of the resource to be located. To this end, node x selects a subset V of U and sends the query to all fingers in V. These fingers will in turn forward the query to all nodes in the portions of the spanning tree they are responsible for, following the DHT broadcast algorithm described above. When a node receives a query, it checks for local items matching the query criteria and, for each matching item, sends a query hit directly to x. The fingers in V are removed from U to indicate that they have been already visited.

After sending the query to all nodes in V, x waits for an amount of time T_L, which is the estimated time needed by the query to reach all nodes, up to a given level L, of the subtrees rooted at the unique fingers in V, plus the time needed to receive a query hit from those nodes. Then, if the current number of received query hits R_c is equal or greater than R_d, x terminates. Otherwise, an iterative procedure takes place.

At each iteration, node x:

1. calculates the item popularity P as the ratio between R_c and the number of nodes already theoretically queried;

2. calculates the number H_q of hosts in the network that should be queried to hit R_d query hits based on P;
3. chooses, among the nodes in U, a new subset V' of unique fingers whose associated subtrees cumulatively contain the minimum number of nodes that is greater than or equal to H_q;
4. sends the query to all nodes in V';
5. waits for an amount of time needed to propagate the query to all nodes in the subtrees associated to V'.

The iterative procedure above is repeated until the desired number of query hits is reached, or there are no more fingers to contact. Note that, if the item popularity is properly estimated after the probe query, only one additional iteration may be sufficient to obtain the desired number of query hits.

An important point in DQ-DHT is estimating the number of nodes present in the different subtrees, and at different levels, of the spanning tree associated with the broadcast process. In the next section we discuss how we calculate such properties of the spanning tree and introduce some functions that are used in the algorithm (described in Section 3.2).

3.1. Properties of the Spanning Tree Associated to the Broadcast Process

As recalled in Section 2, the spanning tree associated with the broadcast over a fully populated Chord ring is a binomial tree. A binomial tree of order $i \geq 0$, B_i, consists of a root with i subtrees, where the j^{th} subtree is a binomial tree of order $j-1$, with $1 \leq j \leq i$. Given a binomial tree B_i, the following properties can be proven [17]: 1) the number of nodes in B_i is 2^i; 2) the depth of B_i is i; 3) the number of nodes at level l in B_i is given by the binomial coefficient $\binom{i}{l}$.

Given the binomial tree properties, we calculate the properties of the spanning tree associated with a broadcast initiated by a node having u unique fingers as shown in Table 1. Basically, we correct the binomial tree properties by a factor $c = N/2^u$, where N is the number of nodes in the network (which can be estimated [18]), to compensate the fact that the value of u may be different from the value of $\log_2 N$ in the case of not fully populated rings. Note that, since the value of D_i may be not an integer, we use the generalized binomial coefficient to calculate N_i^l.

To verify the validity of the formulas defined in Table 1 in the case of not fully populated Chord rings, we employed a custom-made network simulator

Table 1: Properties of the spanning tree rooted at a node with u unique fingers $F_1..F_u$.

Notation	Description	Value
N_i	Number of nodes in the subtree rooted at F_i, where $1 \leq i \leq u$	$2^{i-1} \times c$
D_i	Depth of the subtree rooted at F_i, where $1 \leq i \leq u$	$\log_2 N_i$
N_i^l	Number of nodes at level l of the subtree rooted at F_i, where $1 \leq i \leq u$ and $0 \leq l \leq D_i$	$\binom{D_i}{l}$

(the same used for the performance evaluation presented in Section 4.1). Through the simulator we built several random Chord overlays and compared the real properties of the broadcast spanning tree with the values computed using the formulas in Table 1. The results of an extract of such experiments are presented in Figure 2 and Figure 3.

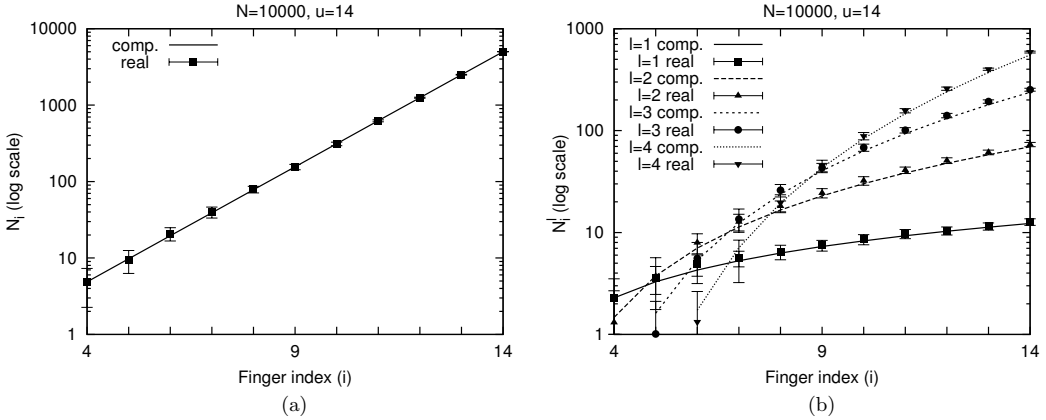


Figure 2: Comparison between computed and real values of N_i and N_i^l for different values of i and l in a Chord DHT with $N = 10000$ from a node with $u = 14$. Lines represent the computed values. Single points with error bars are the real values. The error bars represent the standard deviations obtained from 100 simulation runs.

Figure 2a shows the comparison of computed and real (i.e., measured) values of N_i for different values of i, in a Chord DHT with $N = 10000$ from a node with $u = 14$. As shown by the graph, the means of the real values (represented as points) are very close to the computed values (represented as lines) for any value of i. The graph in Figure 2b considers the same DHT of Figure 2a, but shows the comparison of computed and real values of N_i^l for different values of i, with l ranging from 1 to 4. As before, the mean of the real values resulted very close to the computed values for any value of i and l.

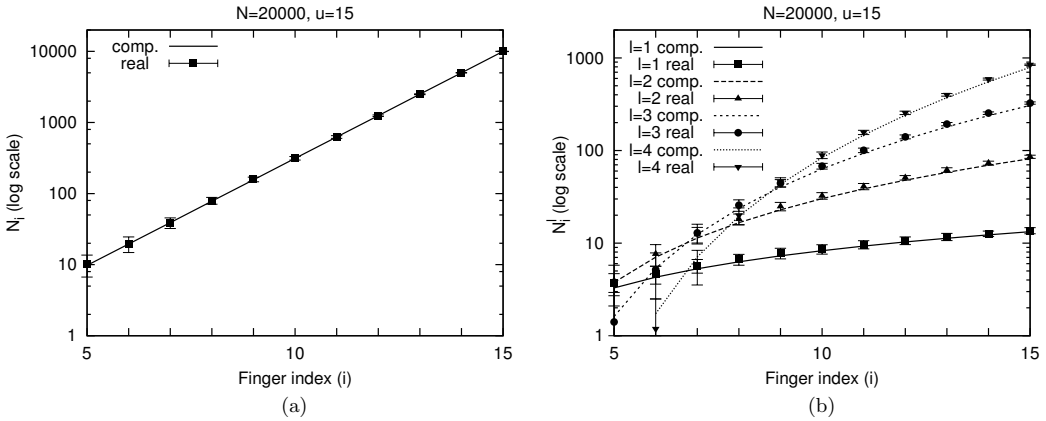


Figure 3: Computed and real values of N_i and N_i^l for different values of i and l in a Chord DHT with $N = 20000$ from a node having $u = 15$.

Figures 3a and Figures 3b show the comparison of computed and real values of N_i and N_i^l in a Chord overlay with $N = 20000$ from a node having $u = 15$. The results are almost identical to those obtained with $N = 10000$ (except for the scale factor), with the computed values very close to the real ones.

In summary, the experimental results presented in Figure 2 and Figure 3 demonstrate that the formulas in Table 1 can also be used to estimate - with high accuracy - the properties of the spanning tree associated with the broadcast process in Chord DHTs.

Based on the spanning tree properties defined in Table 1, we define in Table 2 some aggregate functions operating on a set of unique fingers. Such functions are used in the DQ-DHT algorithm presented in the next section.

3.2. The DQ-DHT Algorithm

DQ-DHT defines two procedures: SubmitQuery, executed by a node to submit a query, and ProcessQuery, executed by a node receiving a query to process.

SubmitQuery (see Fig. 4) receives the query Q and the desired number of results R_d. It makes use of the functions defined in Table 2, and it is assumed that the procedure is executed by a node x.

The procedure starts by initializing to 0 the current number of results R_c (*line 1*). The value of R_c is incremented by 1 whenever a query hit is received. A set U is initialized to contain all unique fingers of node x (*line*

Table 2: Aggregate functions operating on a set V of n unique fingers with indices $i_1..i_n \in [1, u]$.

Function	Returned result	Value
$\mathrm{N}(V)$	Total number of nodes in the subtrees associated with the unique fingers in V	$\sum_{i=i_1..i_n} N_i$
$\mathrm{D}(V)$	Depth of the subtree associated with the unique finger with highest index in V	D_i where $i = \max(i_1..i_n)$
$\mathrm{N}(V, L)$	Total number of nodes from level 0 to level L of the subtrees associated with the unique fingers in V	$\sum_{i=i_1..i_n} \sum_{l=0}^{l_i} N_i^l$ where $l_i = \min(L, D_i)$

2), and H_t is set to $\mathrm{N}(U)$, which corresponds to the total number of hosts that can be queried in the network (*line 3*). The first subset V of fingers to visit is selected from U (*line 4*), and U is updated accordingly (*line 5*).

Afterwards, an integer L between 0 and $\mathrm{D}(V)$ is chosen (*line 6*). The value of L represents the last level of the subtrees associated with V from which to wait a response before estimating the item popularity. The amount of time T_L needed to receive a response from those levels is then calculated as $T_H \times (L + 2)$, where T_H is the average time to pass a message from node to node (*line 7*). The value $L + 2$ is obtained by counting one hop to pass the message from x to the fingers, L hops to propagate the message up to level L, and an additional hop to return the query hit to node x.

Then, Q is sent to all fingers in V invoking the subroutine SEND described below (*line 8*). After the wait (*line 9*), the number of nodes visited H_v is initialized to $\mathrm{N}(V, L)$ (*line 10*). While the popularity will be estimated considering only levels from 0 to L, the query continues to be forwarded up to level $\mathrm{D}(V)$. The additional amount of time T_r that would be necessary to get a response from the remaining levels is therefore proportional to $\mathrm{D}(V) - L$ (*line 11*).

After this first phase, an iterative process takes place while $R_c < R_d$ and there are more fingers to visit ($U \neq \emptyset$) (*line 12*). If at least one result has been received, node x estimates the item popularity P (*line 14*), and the estimated number H_d of hosts to obtain R_d results based on P (*line 15*). Otherwise (i.e., $R_c = 0$), H_d is set to $H_t + 1$, meaning that it is likely that *more than* all available hosts must be contacted to hit R_d results (*line 17*).

If $H_d < \mathrm{N}(V)$, it is expected to receive enough results from the fingers that

```
procedure SubmitQuery(Q, R_d)
 1: R_c ⇐ 0
 2: U ⇐ all unique fingers of node x
 3: H_t ⇐ N(U)
 4: V ⇐ a subset of U
 5: U ⇐ U \ V
 6: L ⇐ an integer ∈ [0, D(V)]
 7: T_L ⇐ T_H × (L + 2)
 8: Send(Q, V)
 9: sleep(T_L)
10: H_v ⇐ N(V, L)
11: T_r ⇐ T_H × (D(V) − L)
12: while R_c < R_d and U ≠ ∅ do
13:     if R_c > 0 then
14:         P ⇐ R_c/H_v
15:         H_d ⇐ R_d/P
16:     else
17:         H_d ⇐ H_t + 1
18:     end if
19:     if H_d ≤ N(V) then
20:         sleep(T_r)
21:         H_v ⇐ N(V)
22:         T_r ⇐ 0
23:     else
24:         H_q ⇐ H_d− N(V)
25:         if H_q > N(U) then
26:             V' ⇐ U
27:         else
28:             V' ⇐ subset of U with min. N(V') ≥ H_q
29:         end if
30:         U ⇐ U \ V'
31:         T_V' ⇐ T_H × (D(V') + 2)
32:         Send(Q, V')
33:         sleep(max(T_V', T_r))
34:         H_v ⇐ N(V)+ N(V')
35:         V ⇐ V'
36:         T_r ⇐ 0
37:     end if
38: end while

subroutine Send(Q, V = {F_i1..F_in})
1: for i = i_1 to i_n do
2:     if i < u then
3:         limit ⇐ F_i+1
4:     else
5:         limit ⇐ x
6:     end if
7:     send message M = {x, Q, limit} to node F_i
8: end for
```

Figure 4: The SubmitQuery procedure.

have been already contacted. Note that this may happen only if $L < \mathrm{D}(V)$, because P is estimated on the basis of the results arriving from nodes up to level L of the subtrees associated with V. Thus, only in this case, the search initiator must wait for the additional amount of time T_r (*line 20*). After the wait, the value of H_v is updated to include all nodes in V (*line 21*), and T_r is set to 0 (*line 22*).

Otherwise ($H_d > \mathrm{N}(V)$), the number of nodes to be queried H_q is given by H_d minus the number of nodes already queried (*line 24*). If H_q is greater than the number of nodes available, the new set V' of fingers to visit is set to U (*line 26*). Else, V' is the subset of U with the minimum value of $\mathrm{N}(V')$ which is greater than or equal to H_q (*line 28*). The elements in V' are removed from U (*line 30*), and the time $T_{V'}$ needed to receive response from all levels of the subtrees associated with V' is calculated (*line 31*).

After sending the query to all nodes in V' (*line 32*), x performs a wait (*line 33*), updates the number of hosts visited (*line 34*), and sets V to V' (*line 35*). The waiting time on *line 33* is the maximum between $T_{V'}$ and

T_r, for managing the case in which the time T_r needed to visit the levels remaining from the previous phase is greater than the time $T_{V'}$ needed to receive a response from all levels in V'. As for *lines 19-22*, this may happen only on the first iteration since after that the timeout is always set to be proportional to $\mathrm{D}(V')$, and so $T_r = 0$ (*line 36*).

The subroutine SEND forwards the query Q to a set of unique fingers V. Basically, it implements the procedure executed by a node x to perform a broadcast (see Section 2). The only difference is that we do not send the message to all unique fingers of x, but only to those in V. The message M sent by x to a node y includes the Id of the querying node (x), the query to be processed Q, and the *limit* parameter used to restrict the forwarding space of node y.

PROCESSQUERY (see Fig. 5) is executed by a node y that receives a message M containing the Id of the search initiator x, the query to process Q, and the *limit* parameter.

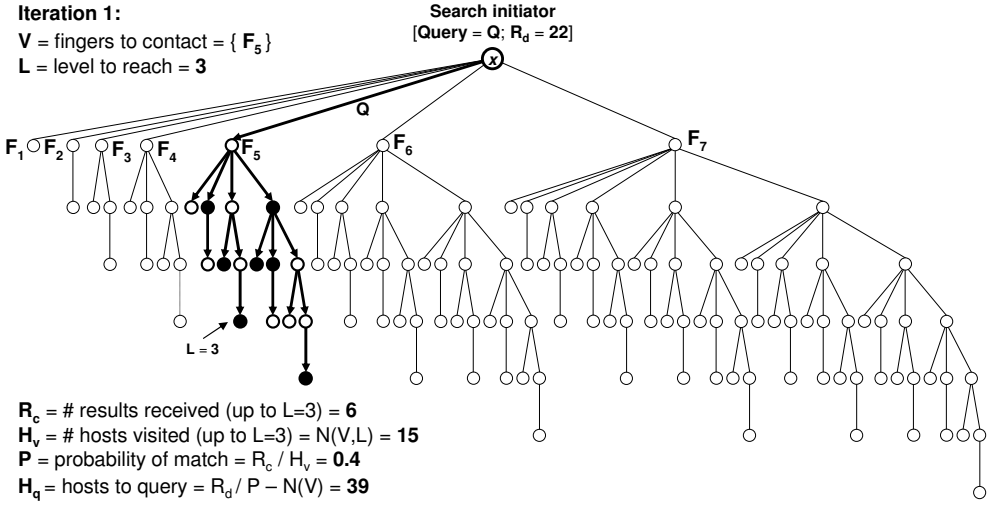
```
procedure ProcessQuery(M = {x, Q, limit})
 1: for i = 1 to u do
 2:     if F_i ∈ ]y, limit[ then
 3:         if i < u then
 4:             oldLimit ⇐ limit
 5:             limit ⇐ F_{i+1}
 6:             if limit ∉ ]y, oldLimit[ then
 7:                 limit ⇐ oldLimit
 8:             end if
 9:         else
10:             limit ⇐ y
11:         end if
12:         send message M = {x, Q, limit} to node F_i
13:     else
14:         exit for
15:     end if
16: end for
17: for each local item matching Q do
18:     send query hit to node x
19: end for
```

Figure 5: The PROCESSQUERY procedure.

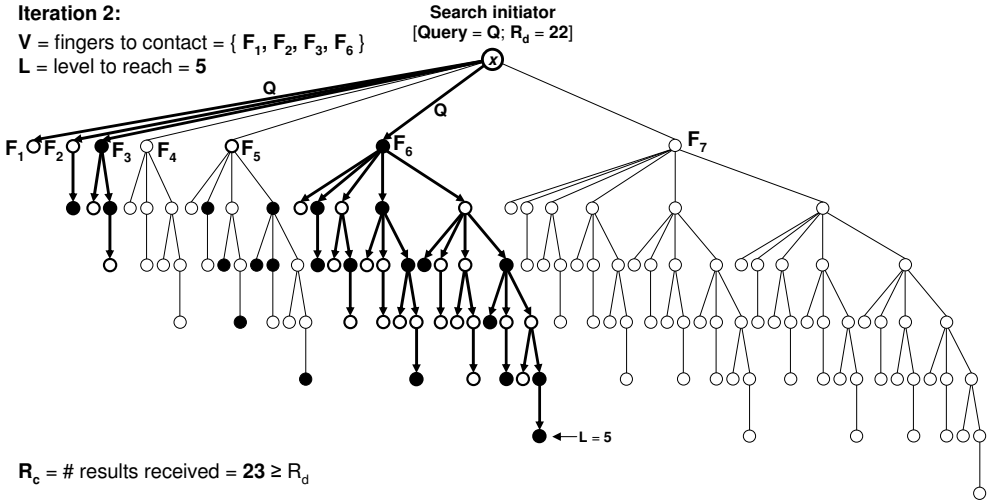
The procedure sends the query to all nodes in the portion of the spanning tree node y is responsible for (*lines 1-16*), following the broadcast algorithm described in Section 2. Then, it processes the query against its local resources, and for each matching item sends a query hit directly to the search initiator (*lines 17-19*).

3.2.1. Example

Figure 6 illustrates an example of a two-iteration DQ-DHT search in a fully populated Chord network with 128 nodes. The root of the binomial tree represents the search initiator, x, and its children represent the fingers $F_1 ... F_7$ of x. In this example the query to process is indicated as Q and the desired number of results is $R_d = 22$.



(a)



(b)

Figure 6: Example of a two-iteration DQ-DHT search in a fully populated Chord network with $N = 128$: (a) iteration 1; (b) iteration 2. Filled black circles are nodes that produced a query hit after receiving the query; empty bold circles are nodes that received the query but did not produce a query hit; empty circles are nodes that did not receive the query.

Figure 6a shows the first iteration of search (probe query) where the following algorithm parameters are used: $V = \{F_5\}$ and $L = 3$ (Section 4.1 discusses how those parameters should be chosen in different scenarios). After

sending Q to F_5, x waits for an amount of time proportional to $L = 3$ before counting the number of results received. The filled black circles represent the nodes, in the subtree rooted at F_5, which have sent a query hit to x in response to Q. Thus, after the first iteration, the number of results is $R_c = 6$. Since $R_c < R_d$, x proceeds by calculating the resource popularity P and the number H_q of hosts to query to obtain a minimum of R_d results, as shown in the figure.

The second iteration is shown in Figure 6b. Given $H_q = 39$, node x chooses the new set of fingers to contact as $V = \{F_1, F_2, F_3, F_6\}$ since the total number of nodes in the subtrees associated with the fingers in V (which is equal to $\sum_{i \in \{1,2,3,6\}} 2^{i-1} = 39$) is the minimum value, greater than or equal to H_q, that can be obtained from the fingers not previously contacted. Then, Q is sent to all fingers in V and, after a waiting period proportional to $L = 5$ (depth of the subtree associated with F_6), the search is concluded because other 17 nodes (filled black circles in the figure) have sent a query hit to x, reaching the goal of obtaining at least R_d results.

4. Performance Evaluation

We evaluate DQ-DHT in terms of two performance parameters: *number of messages* (N_m) and *search time* (T_s). N_m is the total number of messages generated during the search process, while T_s is the amount of time needed to receive the desired number of results.

The system parameters are the number of nodes in the network (N) and the resource replication rate (r), where r is the ratio between the total number of resources satisfying the query criteria and N. The algorithm parameters are the initial set of unique fingers to visit (V), the initial number of levels (L), and the desired number of results (R_d). Even if it is possible to choose V to include an arbitrary subset of the unique fingers of the querying node, we consider the case in which $V = \{F_i\}$, i.e., V includes only the i^{th} unique finger, where $1 \leq i \leq u$. This permits us to have, after the probe query, still $u - 1$ unique fingers from which to choose the new set of subtrees to query, this way improving the granularity of search.

To analyze the message and time complexity of the algorithm we consider the following worst case scenario: at each iteration (including the probe query) the querying node chooses exactly one unique finger to contact, among those not yet contacted. Therefore, the overall search process will complete in u iterations. Since all subtrees are queried one after another, $N_m = N - 1$,

and so the message complexity is $O(N)$. In the same scenario the search time is the sum of the times needed to query all subtrees in sequence, i.e., $T_s = T_H \times \sum_{i=1}^{u}(D_i + 2)$, where T_H is the average time per hop. From Table 1, $D_i = \log N_i = \log(2^{i-1} \times c) = i - 1 + \log c$, where $c = N/2^u$. Thus, $T_s = T_H \times \sum_{i=1}^{u}(i + 1 + \log c) = T_H \times (\frac{1}{2}u^2 + \frac{3}{2}u + (\log c)u)$. Since on average $u = \log N$, we obtain that $T_s = T_H \times (\frac{1}{2}\log^2 N + \frac{3}{2}\log N)$. Therefore, the time complexity in the worst case is $O(\log^2 N)$.

The worst case scenario considered above is based on the very pessimistic assumptions that, at each iteration, the current estimated value of the resource popularity determines the inclusion in the next set V of exactly one unique finger among those still available. In a more typical scenario, assuming a uniform distribution of the matching resources across nodes, the popularity can be estimated with enough accuracy during the probe query, thus allowing most searches to complete in two iterations. In such two-iteration scenario the maximum search time is the sum of the probe query time (which is proportional to $L + 2$) plus the time to cover the deepest subtree that can be chosen for the second iteration (i.e., the subtree associated with F_u): $T_s = T_H \times ((L + 2) + (D_u + 2))$. Since on average $D_u = \log N - 1$, $T_s = T_H \times ((L+2) + (\log N + 1))$ and so the time complexity in such scenario is $O(\log N)$.

4.1. Simulation Analysis

We experimentally evaluated the behavior of DQ-DHT in different scenarios using a custom-made discrete-event simulator written in Java. The tests have been performed in a randomly generated Chord network with N ranging from 10000 to 50000, and a value of r ranging from 0.25 % to 32 %. Moreover, different combinations of the algorithm parameters V, L, and R_d have been experimented. All the results presented in the following are calculated as an average of 100 independent simulation runs, where at each run the search is initiated by a randomly chosen node.

We ran a first set of simulations to evaluate the behavior of DQ-DHT varying the initial set V of unique fingers to contact. To this end, we fixed the number of nodes ($N = 50000$) and the desired number of results ($R_d = 100$). At each run, we chose V to include one of the fingers between F_8 to F_{14}, with the initial value of L set to 5. The graphs in Fig. 7 show number of messages and search time as a function of the replication rate. The search time is expressed in time units, where one time unit corresponds to the average time to pass a message from node to node.

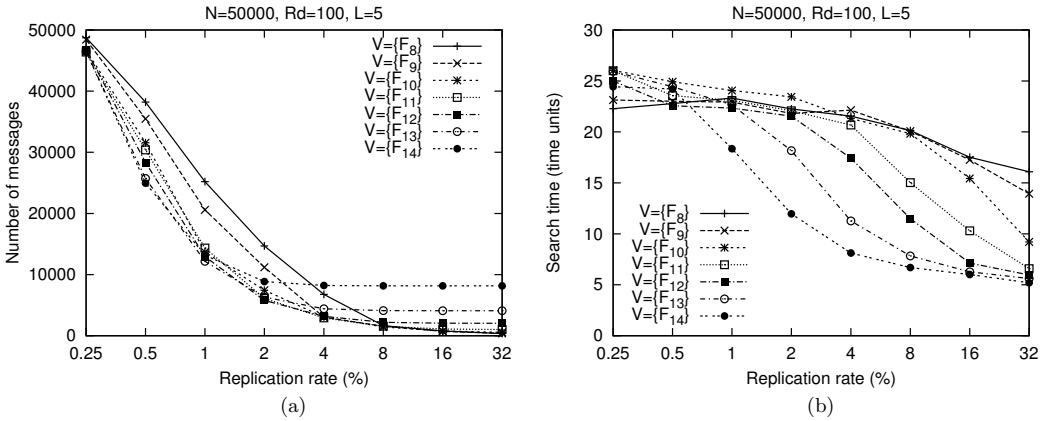


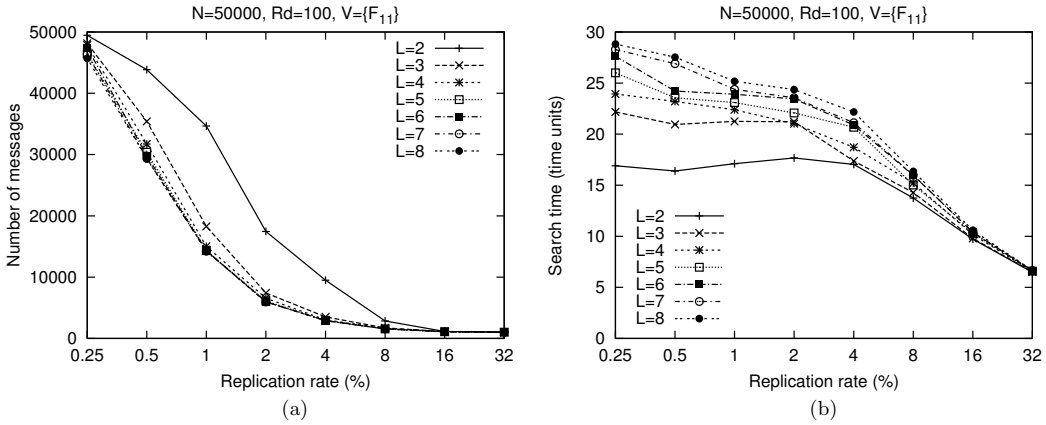
Figure 7: Effect of varying the initial set V, with $L = 5$ and $R_d = 100$, in a network with $N = 50000$: (a) number of messages; (b) search time.

As expected, Fig. 7a shows that the number of messages decreases as the replication rate increases, for any value of V. When $V = \{F_8\}$, the average number of messages passes from 48735 for $r = 0.25\%$, to 360 for $r = 32\%$. In the opposite case, $V = \{F_{14}\}$, the number of messages passes from 46473 for $r = 0.25\%$, to 8159 for $r = 32\%$. For high values of r (i.e., $r = 16 - 32\%$), in most cases the probe query is sufficient to obtain the desired number of results, and so the number of messages corresponds to the number of nodes in the subtree associated with the finger in V.

For values of r lower than 2%, typically at least one additional iteration after the probe query is needed. In these cases, the generated number of messages depends on the accuracy of the popularity estimation, which is better when a higher number of nodes is queried during the probe query (that is, when V includes a finger with a high index). For instance, when $r = 1\%$, the average number of messages is 25207 for $V = \{F_8\}$, 14341 for $V = \{F_{11}\}$, and 13169 for $V = \{F_{14}\}$.

This suggests to start the search by contacting a finger with a high index (e.g., F_{14}), when it is known that the resource is "rare." When there is no information about the popularity of the resource to be found, an intermediate finger (e.g., F_{11}) should be used.

As shown in Fig. 7b, also the search time decreases as the replication rate increases, for any value of V. When $V = \{F_8\}$, the average search time passes from 22.3 for $r = 0.25\%$, to 16.1 for $r = 32\%$. When $V = \{F_{14}\}$, the search time ranges from 24.4 for $r = 0.25\%$, to 5.2 for $r = 32\%$.



(a) (b)

Figure 8: Effect of varying the initial value of L, with $V = \{F_{11}\}$ and $R_d = 100$, in a network with $N = 50000$: (a) number of messages; (b) search time.

The graph shows that with low values of r it is convenient to contact a finger with a high index, which leads to a lower search time with respect to fingers with a lower index. However, since the main objective of DQ-DHT is reducing the number of messages, an intermediate finger (e.g., F_{11}) should be preferred in most cases, even if this may result to an increased search time.

We ran a second set of simulations to evaluate the effect of varying the initial value of L in the same scenario ($N = 50000$ and $R_d = 100$). According to the results discussed above, we chose an intermediate finger for the probe query ($V = \{F_{11}\}$), and varied L from 2 to 8. The results are presented by the graphs in Fig. 8.

Fig. 8b shows that lower values of L generate lower search times. For instance, when $r = 1\%$ the average search time passes from 17.1 with $L = 2$, to 25.2 with $L = 8$. This is mainly due to the fact that the wait after the probe phase is proportional to L, as described in Section 3.2.

On the other hand, Fig. 8a shows that very low values of L produce a significant increase in the number of messages. For example, when $r = 1\%$ the average number of messages passes from 14259 with $L = 8$, to 34654 with $L = 2$. The excess of messages in the second case is due to the reduced accuracy in the estimation of the resource popularity that is obtained considering only a few levels of the subtrees associated with V.

In general, intermediate values of L produce the best compromise between number of messages and search time. For the scenario analyzed here ($V = \{F_{11}\}$), the best result is obtained with $L = 4$, which generates a number of

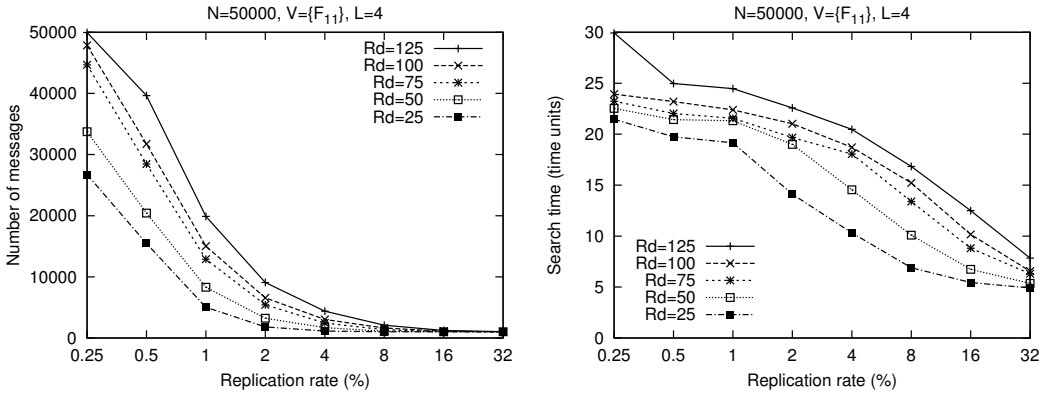


Figure 9: Performance of the algorithm by varying the value of R_d, with $N = 50000$, $V = \{F_{11}\}$ and $L = 4$: (a) number of messages; (b) search time.

messages similar to that produced by higher values of L, but with a quite lower search time, as shown by the graphs in Fig. 8.

A third set of experiments has been performed to evaluate the performance of the algorithm by varying the value of R_d. In particular, fixed $N = 50000$, $V = \{F_{11}\}$ and $L = 4$, we varied the desired number of results from 25 to 125 and measured the number of messages and search time as a function of the replication rate. The results are reported in Fig. 9.

As shown in Fig. 9a, the number of messages is almost proportional to the value of R_d for replication rates greater than or equal to 1%. For example, when $r = 1\%$, the average number of messages is 19884 for $R_d = 125$ and 5021 for $R_d = 25$. With very low replication rates (i.e., $r \leq 0.5\%$), the number of messages still decreases with lower values of R_d, but less proportionally. This is due to the fact the popularity estimation is less accurate, causing more nodes than needed to be contacted. For example, when $r = 1\%$ the average number of messages is 49964 for $R_d = 125$ and 26736 for $R_d = 25$.

Fig. 9b shows the search time obtained in the same scenarios. It can be observed that it decreases significantly when the value of R_d decreases. For example, when $r = 4\%$, the average search time passes from 20.5 for $R_d = 125$ to 10.3 for $R_d = 25$. This confirms the good performance of the algorithm taking also into account that, independently from the desired number of results, there is a fixed cost of six time units $(L+2)$ that are spent to perform the probe querying phase.

Finally, we performed a set of simulations to evaluate the behavior of

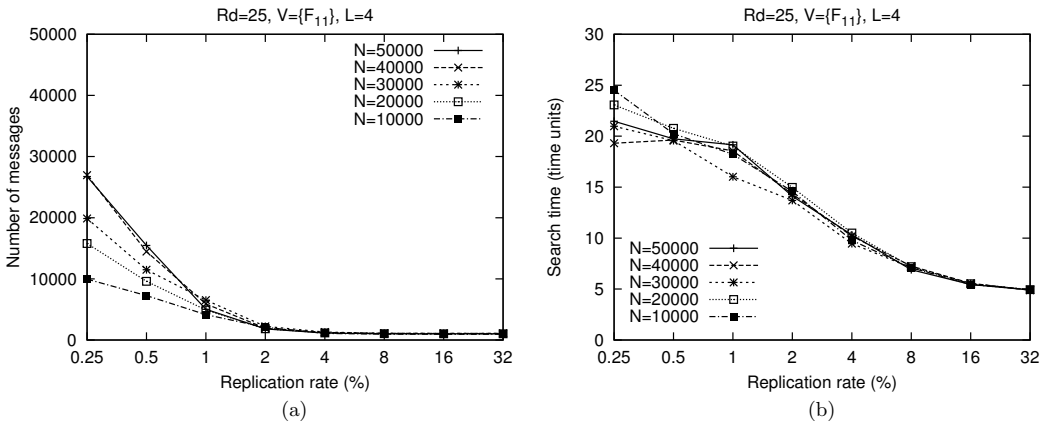


Figure 10: Performance of the algorithm by varying the value of N, with $R_d = 25$, $V = \{F_{11}\}$ and $L = 4$: (a) number of messages; (b) search time.

DQ-DHT by varying the size of the network. In particular, we varied the value of N from 10000 to 50000, fixing $R_d = 25$, $V = \{F_{11}\}$ and $L = 4$. Note that the value of R_d chosen is equal to the number of matching resources that are available in the smallest network ($N = 10000$) with the lowest replication rate ($r = 0.25\%$).

Fig. 10 shows how the number of messages and search time vary as a function of N. Ideally, both the number of messages and the search time should be independent from the network size since the number of nodes to be contacted depends only from R_d and from the replication rate. Fig. 10b shows that the search time is actually independent from the network size, for any value of r.

Fig. 10a shows that also the number of messages is independent from the network size, except for very low replication rates. As highlighted earlier, this is due to the fact that the popularity estimation is less accurate when $r \leq 0.5\%$, causing more nodes than needed to be contacted. Since the number of messages in DQ-DHT cannot be greater than N, this overshooting of the search space is more evident in larger networks.

Based on the results and the analysis presented above, we conclude this section by highlighting which criteria should be followed to choose V and L to maximize the algorithm performance.

- When it is known that the resource to be found is rare, V should include a finger with a high index (e.g., F_{14}). The relatively large number of

messages generated during the probe query will be compensated by the higher accuracy in the popularity estimation, which in turn will contribute to limit the amount of messages generated during the next rounds of search.

- In the absence of information about the popularity of the resource to be found, V should include a finger with a lower index (e.g., F_{11}), which avoids overshooting of the search space during the probe phase, allowing at the same time a good estimation of the resource popularity in most scenarios.

- The value of L should be chosen taking into account that higher values produce higher delays, while lower values produce lower accuracy and therefore more messages. The experimental results showed that intermediate values (e.g., $L = 4$) provide a good compromise between number of messages and search time.

It is worth noticing that the number of nodes to be contacted during the probe phase is determined (through the choice of V and L) only on the basis of the desired accuracy in the popularity estimation. Therefore, the criteria mentioned above should be considered as independent from both the desired number of results and the network size. Indeed, the results presented in Figs. 9 and 10 confirm that the algorithm performs well under various combinations of N and R_d values, using parameters chosen according to the criteria above.

4.2. Comparison with Dynamic Querying in Unstructured Networks

In this section we compare the performance of DQ-DHT with that of DQ in unstructured networks. Since DQ-DHT is designed to work on a DHT-based network, while DQ works on unstructured networks, we adopted the following approach to compare the two systems. First, we built a random Chord network with $N = 50000$ nodes, and measured the average number of unique fingers across all nodes, which resulted to be $\bar{u} = 15.94$. Then, we built an unstructured overlay among the same nodes, in which each node is connected to $\bar{u}$ other random nodes, on the average.

As before, we measured the *number of messages* and the *search time*. In addition, we evaluated the following performance parameters: *duplication rate*, defined as the percentage of duplicate messages on the total number of

messages and *success rate*, defined as the percentage of successful searches on the total number of searches performed.

For DQ in unstructured networks, we implemented the DQ+ algorithm proposed by Jiang and Jin in [19], which is an enhanced version of the original algorithm proposed by Fisk in [13]. The main difference between DQ+ and the original DQ algorithm is briefly described in the following.

In the original DQ, after each iteration, the querying node calculates the total number H_t of hosts to query to reach the desired number of results. Then, it calculates the number H_n of hosts to query per neighbor as H_t/n, where n is the number of neighbors that have not yet received the query. Finally, it calculates the minimum TTL to reach H_n hosts through the next neighbor and sends the query towards that neighbor.

DQ+ adopts a "greedy" strategy. After each iteration, the querying node estimates the total number H_t of host to query, and then calculates the minimum TTL to reach H_t hosts via the next neighbor alone. To avoid overshooting of the search space, DQ+ uses a confidence interval method to estimate the popularity of the searched item. The simulation results presented in [19] show that DQ+ reduces the latency by more than four times with respect to the original DQ algorithm.

The initial parameters of DQ+ are the number of neighbors contacted during the probe phase, n, and the TTL used for the probe query, t. We experimented two configurations: *i*) $n = 3$ and $t = 2$; *ii*) $n = 3$ and $t = 3$. The former is the configuration suggested in [13]; the latter is a variation of the former aimed at increasing the number of nodes contacted during the probe phase, in order to improve the accuracy of the resource popularity estimation. In both cases, the maximum value of TTL allowed after the probe query is 5.

For DQ-DHT we chose the following configurations: *i*) $V = \{F_{14}\}$ and $L = 5$; *ii*) $V = \{F_{11}\}$ and $L = 4$. The first configuration aims at minimizing the search time, but at the cost of a higher number of messages. The second configuration provides a better balance between number of messages and search time, as discussed above.

The results of the comparison between DQ-DHT and DQ+ are presented in Fig. 11. The simulations have been conducted with a value of r ranging from 0.25 % to 32 %, and R_d fixed to 100. Moreover, each result is obtained as the average of 100 independent simulation runs.

As shown in Fig. 11c, the success rate for replication rates greater than or equal to 0.5% is 100% with both DQ+ and DQ-DHT. However, for $r = 0.25\%$,

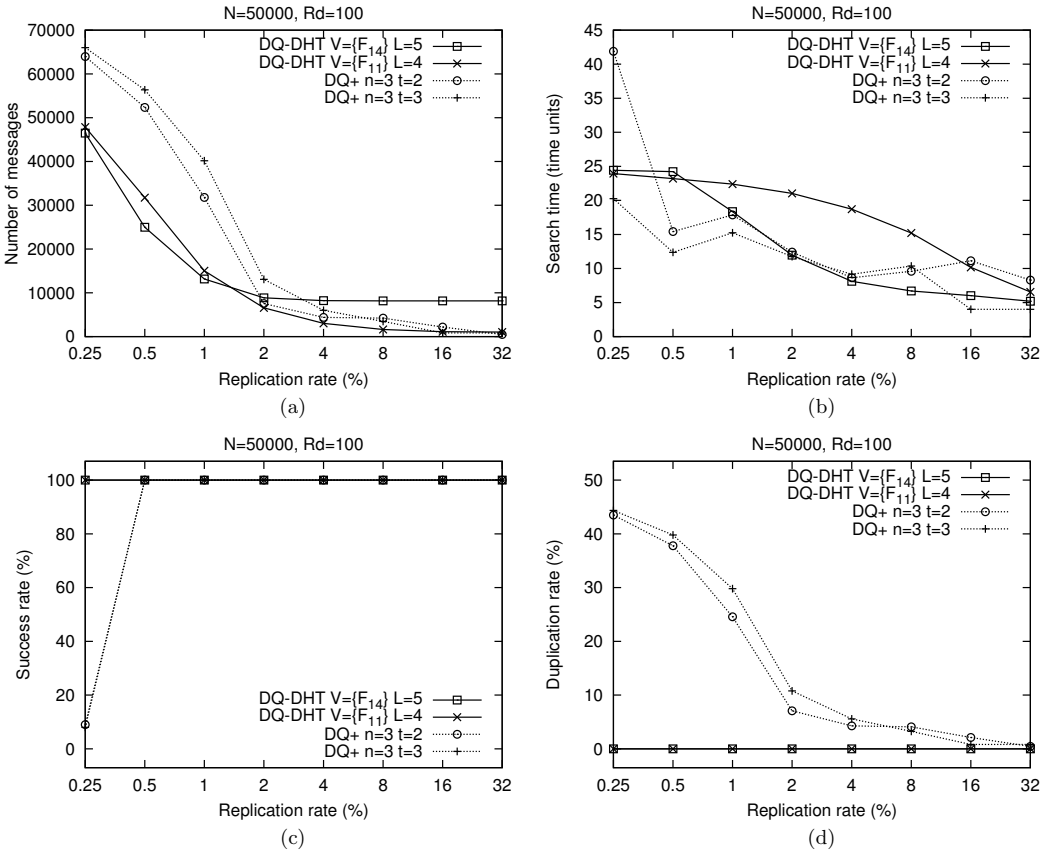


Figure 11: Comparison between DQ-DHT and dynamic querying in unstructured networks (DQ+) with $N = 50000$ and $R_d = 100$: (a) number of messages; (b) search time; (c) success rate; (d) duplication rate.

DQ-DHT has a success rate of 100%, while DQ+ has a success rate of only 8.5%. This is due to the incomplete network coverage of the constrained flooding implemented by DQ+, which in some cases fails to find the desired number of results even if they are actually available in the network. On the contrary, DQ-DHT ensures a complete network coverage and therefore maintains a success rate of 100% even in presence of very low replication rates.

The search time in DQ+ and DQ-DHT is compared in Fig. 11b. As already discussed above, the search time in DQ-DHT strongly depends on the choice of the initial set V. With $V = \{F_{14}\}$ the search time of DQ-DHT is comparable with (and in some cases better than) the search time of

DQ+. This is obtained at the cost of more messages than the case in which $V = \{F_{11}\}$, but they are much less than those generated by DQ+ for values of $r < 2\%$.

Fig. 11a shows the number of messages generated by the two algorithms. DQ-DHT with $V = \{F_{11}\}$ produces less messages than DQ+ for all values of $r \leq 16\%$, while they generate approximatively the same number of messages for $r = 32\%$. For values of r lesser than 2% DQ-DHT outperforms DQ+ by more than a factor of 2. For instance, for $r = 1\%$ DQ-DHT with $V = \{F_{11}\}$ generates 15025 messages, while DQ+ with $t = 3$ produces 40155 messages.

Note that, for low replication rates, DQ-DHT generates less messages with $V = \{F_{14}\}$, while it works better with $V = \{F_{11}\}$ for high replication rates. This is due to the fact that with $V = \{F_{14}\}$ the minimum number of messages sent to the network is higher, and so more messages than needed are generated when the resource to be found is popular. On the other hand, a high number of messages ensures a better accuracy in the estimation of the resource popularity, leading to less messages when the resource to be found is rare.

The greater number of messages generated by DQ+ with respect to DQ-DHT is mainly due to the message duplication caused by flooding. The percentage of duplicate messages on the total number of messages is shown in Fig. 11d. As expected, the duplication rate of DQ+ increases as the replication rate decreases, reaching approximatively the value of 44% with $r = 0.25\%$. DQ-DHT does not suffer the message duplication problem, as each node receives the query at most once. Therefore, the duplication rate for DQ-DHT is 0% for any value of r.

To better evaluate the behavior of DQ-DHT when the desired number of results is lower, we ran a second set of simulations to compare DQ-DHT and DQ+ by reducing the value of R_d to 50. Fig. 12 shows the result of such comparison in terms of number of messages and search time. For the sake of space, we do not include graphs about success rate and duplication rate, which are briefly discussed below. By comparing Fig. 12a-b with Fig. 11a-b, we observe that the relative performances of the two algorithms are similar to those obtained with $R_d = 100$, both in terms of number of messages and search time.

In terms of success rate, DQ+ reaches in this case the same performance of DQ-DHT (100% of success). This is due to the fact that, being the desired number of results lower, DQ+ can find enough relevant nodes even if its flooding strategy does not guarantee complete network coverage. On the

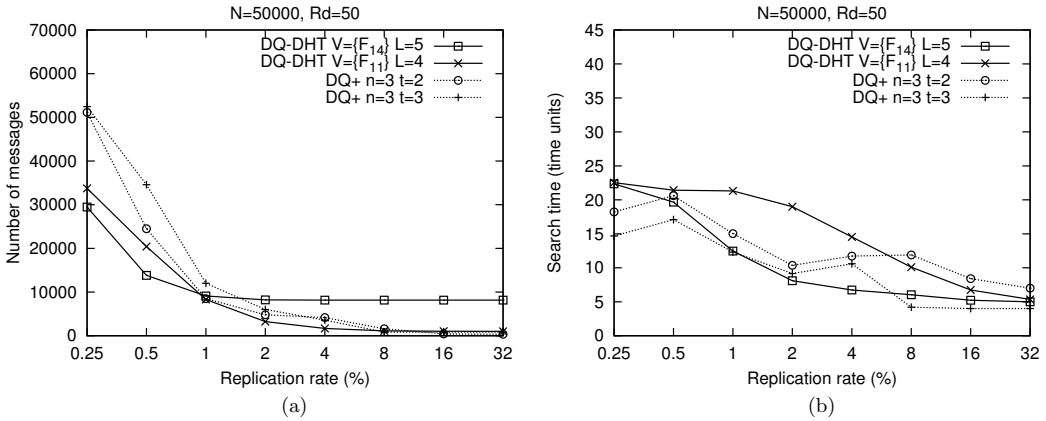


Figure 12: Comparison between DQ-DHT and DQ+ with $N = 50000$ and $R_d = 50$: (a) number of messages; (b) search time.

other hand, DQ-DHT still outperforms DQ+ in terms of duplication rate, which reaches the value of 38% with DQ+, while it is always 0% with DQ-DHT.

In summary, the simulation results presented throughout this section show that DQ-DHT produces much less network overhead (i.e., number of messages) than DQ+, with a comparable - and in some cases better - search time, and with a higher success rate when the resource to be found is rare.

5. Related Work

The work most related to DQ-DHT is the Structella system designed by Castro et al. [20]. Structella replaces the random graph of Gnutella with the structured overlay of Pastry [3], while retaining the content placement and discovery mechanisms of unstructured P2P systems to support complex queries. Two discovery mechanisms are implemented in Structella: constrained flooding and random walks.

Constrained flooding is based on the algorithm of broadcast over Pastry presented in [21]. A node x broadcasts a message by sending the message to all the nodes y in the Pastry's routing table. Each message is tagged with the routing table row r of node y. When a node receives a message tagged with r, it forwards the message to all nodes in its routing table in rows greater than r. To constrain the flood, an upper bound is placed on the row number of entries to which the query is forwarded.

Random walks in Structella are implemented by walking along the ring formed by neighboring nodes in the identifier space. When a node receives a query in a random walker, it uses the Pastry's leaf set to forward the query to its left neighbor in the identifier space. It also evaluates the query against the local content and sends matching content back to the query originator. A random walker is terminated when it finds matching content. Multiple concurrent random walkers can be used to improve search time.

DQ-DHT and Structella share the same goal of supporting complex queries in structured networks. However, DQ-DHT has been designed to find an arbitrary number of resources matching the query criteria, while Structella is designed to discover just one of such resources. In Structella in fact, with both constrained flooding and random walks, a node stops forwarding a query if it has matching content. This functional difference makes DQ-DHT and Structella not comparable, so we cannot provide a comparison of their performance.

A way to let Structella return an arbitrary number of results instead of just one could be modifying its random walks algorithm, using the same termination mechanisms proposed for random walks in unstructured networks [22]. Unfortunately, a direct interaction between querying node and walkers may be infeasible in some networks (e.g., due to firewalls) and generates overload of the querying node if too many walkers are used or the communication with them is too frequent [22]. It is worth noticing that DQ-DHT, on the contrary, does not require any remote interaction to terminate the search.

A few other research works broadly relate to our system for their combined use of structured and unstructured P2P techniques.

Loo et al. [23] propose a hybrid system in which DHT-based techniques are used to index and search rare items, while flooding techniques are used for locating highly replicated content. Search is first performed via conventional flooding techniques of the overlay neighbors. If not enough results are returned within a predefined time, the query is reissued as a DHT query. This allows fast searches for popular items and at the same time reduces the flooding cost for rare items.

A critical point in such system is identifying which items are rare and must be published using the DHT. Two techniques are proposed. A first heuristic classifies as rare the items that are seen in small result sets. However, this method fails to classify those items that have not have been previously queried and found. Another proposal is to base the publishing on

well-known term frequencies, and/or by maintaining and possibly gossiping historical summary statistics on item replicas.

Another example is the work by Zaharia and Keshav [24], who focus on the problem of selecting the best algorithm to be used for a given query in a hybrid network allowing both unstructured search and DHT-based lookups. A gossip-based algorithm is used to collect global statistics about document availability and keyword popularity that allow peers to predict the best search technique for a given query.

Each peer starts by generating a synopsis of its own document titles and keywords and labels it as its "best" synopsis. In each round of gossip, it chooses a random neighbor and sends the neighbor its best synopsis. When a node receives a synopsis, it fuses this synopsis with its best synopsis and labels the merged synopsis as its best synopsis. This results in every peer getting the global statistics after $O(\log N)$ rounds of gossip.

Given a query composed of a set of keywords, a peer estimates the expected number of documents matching such a set of keywords using the information in its best synopsis. If this number is over a given threshold, many matches are expected, so the peer floods the query. Otherwise, it uses the DHT to search for each keyword, requesting an in-network join, if that is possible. The flooding threshold is dynamically adapted by computing the utility of both flooding and DHT search for a randomly chosen set of queries.

It is worth noticing that the last two systems do not support arbitrary queries since information about resources is published and searched using DHT-based mechanisms. DQ-DHT, on the contrary, supports arbitrary queries in an easy way since content placement is unrelated to the DHT overlay and query processing is performed on a node-by-node basis.

6. Conclusions

A way to support arbitrary queries in structured networks is implementing unstructured search techniques on top of DHT-based overlays. Following this approach, we proposed DQ-DHT: a P2P search algorithm that combines the dynamic querying technique with an algorithm for efficient broadcast over a DHT. DQ-DHT has been particularly designed to be used in large-scale distributed systems, like computational grids, where it is necessary to support arbitraries queries for searching resources on the basis of complex criteria or semantic features.

The behavior of DQ-DHT has been analyzed through a simulator by varying its initial configuration, in order to understand which are the best parameters to use based on user/system requirements and objectives (i.e., minimizing the number of messages or the search time). We also compared the performance of DQ-DHT with that of the enhanced dynamic querying in unstructured networks (DQ+). The simulation results show that DQ-DHT generates much less network overhead than DQ+, with a comparable (and in some cases better) search time, and with a higher success rate when the resource to be found is rare.

References

[1] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, S. Haridi, Peer-to-Peer Resource Discovery in Grids: Models and Systems. Future Generation Computer Systems 23(7) (2007) 864-878.

[2] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, H. Balakrishnan, Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. ACM SIGCOMM'01, San Diego, USA, 2001.

[3] A. Rowstron, P. Druschel, Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. Middleware 2001, Heidelberg, Germany, 2001.

[4] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, J. Kubiatowicz, Tapestry: a resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications 22(1) (2004) 41-53.

[5] M. Ripeanu, A. Iamnitchi, I. Foster, Mapping the Gnutella Network. IEEE Internet Computing 6(1) (2002) 50-57.

[6] N. Leibowitz, M. Ripeanu, A. Wierzbicki, Deconstructing the Kazaa network. 3rd IEEE Workshop on Internet Applications (WIAPP 2003), San Jose, USA, 2003.

[7] A. Andrzejak, Z. Xu. Scalable, Efficient Range Queries for Grid Information Services, 2nd IEEE Int. Conf. on Peer-to-Peer Computing (P2P 2002), Linköping, Sweden, 2002.

[8] M. Cai, M. R. Frank, J. Chen, P. A. Szekely, MAAN: A Multi-Attribute Addressable Network for Grid Information Services. Journal of Grid Computing 2(1) (2004) 3-14.

[9] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, I. Stoica, Complex Queries in DHT-based Peer-to-Peer Networks. 1st Int. Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, USA, 2002.

[10] M. Castro, M. Costa, A. Rowstron, Debunking Some Myths About Structured and Unstructured Overlays. 2nd Symp. on Networked Systems Design and Implementation (NSDI'05), Boston, USA, 2005.

[11] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker, Making Gnutella-like P2P Systems Scalable. ACM SIGCOMM'03, Karlsruhe, Germany, 2003.

[12] S. El-Ansary, L. Alima, P. Brand, S. Haridi, Efficient Broadcast in Structured P2P Networks. 2nd Int. Workshop on Peer-to-Peer Systems (IPTPS'03), Berkeley, USA, 2003.

[13] A. Fisk, Gnutella Dynamic Query Protocol v0.1, 2003. http://www9.limewire.com/developer/dynamic_query.html.

[14] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. Journal Supercomputer Applications 15(3) (2001).

[15] D. Talia, P. Trunfio, Dynamic Querying in Structured Peer-to-Peer Networks. 19th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM 2008), Samos Island, Greece, LNCS 5273, pp. 28-41, 2008.

[16] J. C. Y. Chou, T.-Y. Huang, K.-L. Huang, T.-Y. Chen, SCALLOP: A Scalable and Load-Balanced Peer-to-Peer Lookup Protocol. IEEE Trans. Parallel Distrib. Syst. 17(5) (2006) 419-433.

[17] B. R. Preiss, Data Structures and Algorithms with Object-Oriented Design Patterns in C++. John Wiley & Sons, 1998.

[18] A. Binzenhöfer, D. Staehle, R. Henjes, Estimating the size of a Chord ring. Technical Report 348, Institute of Computer Science, University of Würzburg, 2005.

[19] H. Jiang, S. Jin, Exploiting Dynamic Querying like Flooding Techniques in Unstructured Peer-to-Peer Networks. 13th IEEE Int. Conf. on Network Protocols (ICNP 2005), Boston, USA, 2005.

[20] M. Castro, M. Costa, A. Rowstron, Should we build Gnutella on a structured overlay? Computer Communication Review 34(1) (2004) 131-136.

[21] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, A. Wolman, An Evaluation of Scalable Application-Level Multicast Built Using Peer-to-Peer Overlays. IEEE INFOCOM'03, San Francisco, USA, 2003.

[22] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker, Search and Replication in Unstructured Peer-to-Peer Networks. 16th ACM Int. Conf. on Supercomputing, New York, USA, 2002.

[23] B. T. Loo, R. Huebsch, I. Stoica, J. M. Hellerstein, The Case for a Hybrid P2P Search Infrastructure. 3rd Int. Workshop on Peer-to-Peer Systems (IPTPS'04), La Jolla, USA, 2004.

[24] M. Zaharia, S. Keshav, Gossip-based Search Selection in Hybrid Peer-to-Peer Networks. 5th Int. Workshop on Peer-to-Peer Systems (IPTPS'06), Santa Barbara, USA, 2006.