UNIVERSITÀ CA' FOSCARI DI VENEZIA
Dipartimento di Informatica
Technical Report Series in Computer Science

Rapporto di Ricerca CS-2006-6

Luglio 2006

M. Marzolla, M. Mordacchini, S. Orlando

Peer-to-Peer Systems for Discovering Resources in a Dynamic Grid

Dipartimento di Informatica, Università Ca' Foscari di Venezia
Via Torino 155, 30172 Mestre–Venezia, Italy

Peer-to-Peer Systems for Discovering Resources in a Dynamic Grid

Moreno Marzolla [b] Matteo Mordacchini [a,b,*] Salvatore Orlando [a,c]

[a] *Dip. di Informatica, Univ. Ca' Foscari di Venezia, via Torino 155, 30172 Mestre, Italy*
[b] *INFN Sezione di Padova, via Marzolo 8, 35131 Padova, Italy*
[c] *ISTI Area della Ricerca CNR, via G. Moruzzi 1, 56124 Pisa, Italy*

Abstract

The convergence of the Grid and Peer-to-Peer (P2P) worlds has led to many solutions that try to efficiently solve the problem of resource discovery on Grids. Some of these solutions are extensions of P2P DHT-based networks. We believe that these systems are not flexible enough in case the indexed data are very dynamic, i.e., the values of the resource attributes change very frequently over time. This is a common case for some data managed by typical Grid systems, like CPU loads, queue occupation, etc. Moreover, since common requests for Grid resources may be expressed as multi-attribute range queries, we think that the DHT-based P2P solutions that have been proposed so far with the aim of supporting such type of queries can suffer from poor flexibility and efficiency.

In this paper we thus present a couple of P2P systems. Both the systems are based on Routing Indexes, used to efficiently route queries and update messages in the presence of highly variable data. The first system needs the adoption of a tree-shaped overlay network. The second one, which is an evolution of the first, is based on a two-level hierarchical network topology, where tree topologies must only be maintained at the lower level of the hierarchy, i.e., within the various node groups making up the network. The main goal of the second organization is to achieve a simpler maintenance of the overall P2P graph topology, by preserving the good properties of the tree-shaped topology.

We discuss the results of extensive simulation tests aimed at assessing the performance and scalability of the proposed approaches, by also studying how the network topologies affect the propagation of query and update messages.

Key words: Peer-to-Peer, Multi-attribute Range Queries, Routing Indexes, Overlay Networks, Resource Discovery, Grid Computing.

1 Introduction

P2P networks have emerged as one of the most successful ways to share resources (e.g., data, storage, computational power) in a distributed, scalable, and fault tolerant way. Since large scale resource sharing is also the goal of modern Grid systems, P2P systems and Grid worlds are slowly converging (Foster and Iamnitchi, 2003; Talia and Trunfio, 2003), leading to application of P2P techniques to Grid systems.

One of the core functionalities of Grid systems is the location of resources satisfying given constraints. This occurs when a Grid user submits a job, and specifies its requirements, like memory, disk space, Operating System version, etc. Locating data that match a given search criteria is one of the most studied problems in P2P systems. However, the Grid resource location problem is more complex, as resource features may be *dynamic*. For example, the available disk space available at a given storage element varies over time as users add and remove data. Location of dynamic data on distributed P2P systems is considerably more difficult than location of static data. Moreover, Grids users are interested in finding resources that match *multi-attribute range queries*, i.e. queries that identify all (or a subset of) the resources characterized by a set of attributes whose values fall into given intervals. Note that this is different from typical file sharing P2P systems, which usually index and manage *(key, values)* pairs, and naturally support exact queries for a value (e.g., File Location) given a search key.

A request for locating and acquiring resources in order to execute a job is a typical case of multi-attribute range query. Consider the following example (Pacini, 2005):

$$\begin{aligned} Q = \{R \in \{R_1, \dots R_N\} \mid & \mathit{CpuSpeed}[R] \geq 2.0\mathit{GHz} \\ & \textbf{and}\, \mathit{RamSize}[R] \geq 512\mathit{MB} \\ & \textbf{and}\, \mathit{Utilization10}[R] \leq 0.3 \\ & \textbf{and}\, 100\mathit{MB} \leq \mathit{Free\_Space}[R] \leq 300\mathit{MB}\} \end{aligned}$$

which is a query that looks for computational resources R with CPU speed at least $2.0\mathit{GHz}$, at least 512 MB of RAM, with utilization over the last 10 minutes of at most 0.3, and with available disk space in the range $[100, 300]$ MB. Note that the *Utilization* parameter is a typical dynamic attribute, which may vary over time.

$\star$ This work has been partially supported by the European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies (CoreGRID)

$*$ Corresponding Author

Email addresses: marzolla@pd.infn.it (Moreno Marzolla), mordacchini@dsi.unive.it (Matteo Mordacchini), orlando@dsi.unive.it (Salvatore Orlando).

In the following we thus assume that a generic query predicate is a boolean composition of range conditions on resource attributes, and discuss the proposal for novel P2P systems able to locate such resources. It is worth noting that, for expressing range queries, it is necessary to define a total ordering over the domain of resource attribute values. Such total ordering is implied for many attribute types whose values are numbers or strings.

The first unstructured P2P systems that tried to solve the data location problem flood the entire network until all the desired data are collected or a stop condition is reached. This behavior implies a large network traffic overhead which seriously limits the scalability of the system. In order to address this problem more efficiently, many data indexing systems and more structured P2P networks have been proposed, such as the Distributed Hash Table (DHT)-based ones (Balakrishnan et al., 2003; Stoica et al., 2003; Ratnasamy et al., 2001; Rowstron and Druschel, 2001). These systems only allow exact queries to be expressed, while one of the common way to retrieve data on Grids is through *multi-attribute range queries*. Some authors have thus proposed extensions to the cited algorithms in order to adapt these DHT systems to the Grid needs (Cai et al., 2003; Andrzejak and Xu, 2002; Bharambe et al., 2004).

Despite the good results achieved by DHT networks in several fields, they may not represent the best solution in presence of dynamic data, i.e., data whose value change frequently and unpredictably over time. The main problem with DHT-based networks is related to the fact that each change in the resource values requires to re-index the items whose content have changed. Moreover, the DHT-based solutions that have been proposed so far to support multi-attribute range queries can suffer from poor flexibility and efficiency.

We believe that the P2P networks that are less structured than the DHT ones are more suitable to deal with dynamic data and such kind of queries, and thus in this paper we propose a couple of P2P systems based on Routing Index (RI) (Crespo and Garcia-Molina, 2002). To this end, we assume that each P2P node of our network manages and indexes information associated with a disjoint subset of all the Grid resources. Each node also maintains, for each attribute that characterizes the managed resources, a bitmap index, i.e., a condensed description of the local presence/absence of resources. We use bitmap indices not only to represent local resources, but also as a condensed description of the resources present in every neighbors with respect to the overlay network. The last bitmap indexes are thus used as a sort of RI (Crespo and Garcia-Molina, 2002) to route queries towards the location of resources possibly satisfying the query. Thanks to their simplicity, such the indices can easily be updated if some attribute value changes. The updates are propagated from the node responsible for a given resources by gossiping. Such updates are flooded by using the same overlay network exploited for routing the queries.

The first proposal we discuss deals with *Tree Vector*, a P2P discovery system based on a tree-structured overlay network. This allows the bitwise routing indices to represent the complete knowledge about the resources that are reachable by following a link that is connected to a given subtree of nodes. The aim of *Tree Vector* is to build a P2P system with a reasonable trade-off between the need of efficiently route range queries, and the ability to reduce the overhead needed to modify the indices when data changes over time.

Our second proposal is an evolution of *Tree Vector*, and is based on a new topology organization of the overlay network. This new organization, called *Forest of Trees*, aims to ease the maintenance of the network, while preserving the good features of the tree-shaped P2P overlay network. In order to avoid the cost and difficulty in maintaining a very large tree topology, we thus propose a two-level hierarchical network topology, where tree topologies must only be maintained at the lower level of the hierarchy. Since the upper level is instead completely unstructured, this may introduce some imprecision in the RIs. The idea is thus to tradeoff a simpler maintenance of the overall P2P network topology, with the introduction of some imprecision in the routing indices built at the upper level of the network hierarchy.

For each of the two proposed solutions, whose preliminary results were presented in (Marzolla et al., 2006b,a), we perform extensive simulation experiments to assess the performance and scalability of the proposed approach. In particular, we study how the network topology affects the propagation of query and update messages, and derive simple analytical expressions, like the precision and the recall of our search algorithm as a function of query selectivity, index size, and network topology.

This article is structured as follow. In Section 2 we review some previous results in the area of resource discovery in P2P networks. In Section 3 we precisely state the problem we address. Section 4 introduces a first solution to the dynamic resource discovery in Grids, based on a tree overlay network built on the set of peers. In Section 5 we partly relax the requirement of having a tree overlay network, to explore a different peer organization based on a forest of trees (individual trees may be arbitrarily connected to other trees). Final remarks and open issues are discussed in Section 6 .

2 Related Works

The problem of routing queries in P2P systems is well known. In order to avoid flooding the network with query messages (as done by systems like Gnutella (Gnutella, 2006)), many data indexing methods have been proposed. The most promising ones are the so-called DHT-based systems. In these systems every data item is associated with a key obtained by hashing an attribute of the object (e.g. its name). Every node in the network is responsible for maintaining information about a set of keys and the associated items. They also maintain a list of adjacent or neighboring nodes. A query becomes the search of a key in the DHT. When a peer receives a query, if it does not have the requested items, it forwards the query to the neighbor having keys which are closer to the requested one. Data placement ensures that queries eventually reach a matching data item.

In order to further enhance the search performance, many DHT-based protocols organize the peers into an overlay structure. So, in Chord (Stoica et al., 2003) nodes are organized into a virtual circle, while in CAN (Ratnasamy et al., 2001) the identifier space is seen as a d-dimensional Cartesian space. This space is partitioned into zones of equal size and every peer is responsible of one of these zones. Other relevant examples of this kind of systems are Pastry (Rowstron and Druschel, 2001) and Tapestry (Zhao et al., 2001). Although these networks show good performances and scalability characteristics, they only support exact queries, i.e., requests for data items matching a given key. Moreover the hashing mechanism works well with static object identifiers like file names, but is not suitable for handling dynamic object contents.

The ability to perform multi-attribute range queries over mutable data stores is a key feature in many scenarios, like distributed database and Grid resource discovery. Range queries are queries that requests all items whose attribute value fall into a given interval. Some systems have been proposed to support range and multi-attribute queries in P2P networks. The P-Tree (Crainiceanu et al., 2004) uses a distributed version of the B^+-tree index structure. Other protocols use locality preserving hash functions, like the Hilbert space-filling curve, to allow DHT to support range queries. For example, in (Andrzejak and Xu, 2002) the authors propose an extension of the Chord protocol to support range and multi-attribute queries, by using a uniform locality preserving hash function to map items in the Chord key space. in (Ganesan et al., 2004) two methods are proposed. The first one (called SCRAP) adopts space filling curves as hash functions. The second one (MURK) partitions the data space into rectangles (hyper-rectangles) of different size, such that the amount of data stored on peers is equally distributed. Another common solution adopted in literature to resolve multi-attribute queries is to maintain a separate DHT layer for each attribute type. This solution is adopted, among others, by (Cai et al., 2003), (Bharambe et al., 2004), and (Spence and Harris, 2003). In (Cai et al., 2003) the authors extend the CAN protocol using the Hilbert space-filling curve and load balancing mechanisms, while in (Bharambe et al., 2004) a Grid P2P extension of the Symphony DHT system is proposed. In both cases, the authors adopt a solution that maintain a separate DHT for each attribute of the resources present in the network, but the nodes of each DHT also store information concerning the other attributes. Then, query routing is performed only in the DHT of the attribute with the lowest selectivity, as it requires the lowest communication cost. Once the resources matching the sub-query of such an attribute are found, the values of the other attributes are checked in order to find the final set of matching resources. In (Spence and Harris, 2003) the authors use an extension of the Pastry DHT network. The system uses one Pastry ring for each attributes. The index portions store not only the values of the resources but also the root values of so-called *Aggregation Points*. These are prefix trees of the resources attribute values (which are the leaves of these trees) and they are used to perform range queries in XenoSearch. The matching results of each sub-query are finally intersected at the query originator node.

Another form of distributed indexes can be supported even on unstructured P2P networks by using the so-called RI (Crespo and Garcia-Molina, 2002). RIs are based on the content of the data present on each node. Each peer in the network maintains both an index of its local resources and a table for every neighbor, which summarizes the data that is reachable trough all the path that start from that neighbor. When a peer receives a query, it checks if the requested items are present locally and then forwards the query to the neighboring node which has, accordingly to the RI, the most relevant data with respect to the query. The process is iterated until a stop condition is achieved (e.g. the desired number of results is reached).

One of the common limitations of many of the techniques proposed in the literature is their inefficiency in maintaining the indexes in the presence of dynamic data. The need for DHT systems to re-index the items whose values have changed may lead to a great amount of overhead when changes happen frequently. Moreover, the advantages of DHT networks on exact queries may be reduced when dealing with range queries, particularly when the requested ranges are sufficiently large. Moreover, some solutions presented for the multi-attribute query case show some scalability problems with respect to the number of attributes, as they use one separateDHT for each attribute type. We try to address all of these limitations in this article: we present a solution to the problem of dynamic data location with multi-attribute, range queries. We use a form of RI in order to achieve a good tradeoff between query routing efficiency and the need to limit updates occurring when some data items change value.

3 Problem Statement

We suppose that each peer in the system holds a (possibly empty) set of data items, also called *local repository*. Each data item is described by a set of attribute-value pairs. For example, in a distributed relational database, a data item would be a database record, and the attribute-value pairs would be the names and corresponding value of the attributes of each table. In a Grid Information System, such data items would model Grid resources, each in turn characterized by a set of features (attribute-value pairs). We also suppose that data items are dynamic, i.e., the value of the attributes may change over time. Users of the P2P system want to locate data items satisfying given search criteria, which are expressed as partial range queries over the set of attributes.

More specifically, we consider a P2P system where each peer implements the following operations:

insert$(D, \{A_1 : V_1, \ldots A_r, V_r\})$ Insert a new data item D on the local repository; the data item has attributes $A_1, \ldots A_r$ with values $V_1, \ldots V_r$ respectively.

update$(D, A : V_{\textbf{new}})$ Change the value of attribute A for data item D on the local repository; the new value will be V_{new}.

lookup(Q) Search for data items matching query Q over the whole P2P system (including the current node).

Additionally, peers may join and leave the system at any time; as usual in P2P systems, we want to rely as few as possible on any centralized information.

In the following we consider a P2P system with a set $P = \{P_1, P_2, \ldots P_N\}$ of N peers. We denote with *Data* (P_i) the local repository on peer P_i. Each data item is labelled with a set of attribute-value pairs. We suppose that there is a limited number of different attribute names. We denote with $\{A_1 : T_1, \ldots A_M : T_M\}$ the set of all the M possible attribute names with their corresponding types. Data types T_i can be any arbitrary data types, subject to the constraint that there must be a total ordering defined over T_i. Each data item can be labelled with any nonempty subset of attributes of $\{A_1, \ldots A_M\}$. For each data item D, we denote with *AttList* (D) the set of all attribute names defined for D. Moreover, for each attribute $A \in$ *AttList* (D), $D[A]$ denotes the value of attribute A for data item D.

The system provides a query facility for locating all data items matching a user-defined partial range query Q. We consider queries generated by the following grammar (we assume that the usual operator precedence rules apply):

$$Q := Q \textbf{ and } Q \mid Q \textbf{ or } Q \mid v_1 \leq A \leq v_2$$
$$A := A_1 \mid \ldots \mid A_M$$

We consider partial range queries over subsets of the attributes, that is, boolean compositions of range predicates $v_1 \leq A \leq v_2$. Multiple conditions over different attributes are possible. Conditions such as $A \leq v_2$, $A \geq v_1$ and $A = v_1$ are special cases of $v_1 \leq A \leq v_2$ which can be expressed by setting $v_1 = -\infty$, $v_2 = +\infty$ and $v_1 = v_2$ respectively.

Then, each query carries the following information:

- Query ID
- Sub-query q_1 on attribute A_1 lower and upper bounds
- $\ldots$
- Sub-query q_n on attribute A_n lower and upper bounds

Observe that the user is not required to specify conditions on all attributes of a data item. The result of a **lookup**(Q) operation is to return the set of all the locations (i.e., the set of peer IDs) of all data items D matching Q.

A trivial way of locating resources in an unstructured P2P network would be to flood the range queries to all nodes within a given radius from the originating peer. This is clearly undesirable, as (1) flooding generates a potentially high message load on all nodes, including those which do not hold resources satisfying the queries; and (2) setting a maximum hop count to stop the query from flooding the entire network does not guarantee that all matches are located.

In order to limit the flooding of queries, we build a specific overlay network over the set of peers, and associate routing information with individual links. In the following we thus discuss a couple of P2P systems that exploit different overlay networks and a particular form of RIs to route queries and updates of attribute values stored in each peer.

We want now to point out some considerations about multi-attribute query resolution. Let Q be a multi-attribute, range query over a atributes, such that $Q = q_1$ *op* $q_2 \ldots$ *op* q_a, where *op* $= \vee$ or $\wedge$. Let p be the probability for a peer to match query Q, p_i the probability to match the sub-query q_i. We can express p as a function of the various p_i's: if *op* $= \wedge$, we have that $p = \prod_{i=1}^{a} p_i$. On the other hand, If *op* $= \vee$, we have that $p = 1 - \prod_{i=1}^{a}(1 - p_i)$. More generally, the match probability p_{ij} of a query in the form $q_i \wedge q_j$ is $p_{ij} = p_i p_j$, where the match probability of a query in the form $q_i \vee q_j$ is $p_{ij} = 1 - (1 - p_i)(1 - p_j)$.

Note that, since the query routing strategy is thus based on a selective flooding based on RIs, and the number of nodes targeted by a query Q depends on p, we can expect similar performances of the system in the presence of the same value of p, although obtained by the combination of different numbers of attributes.

4 Tree Vector: a tree-based P2P discovery system

We start the discussion on our proposals for a new P2P discovery system for Grid resources from *Tree Vector*, which is a system that has to maintain an undirected spanning tree over all the set P of peers. In this system, the tree-shaped overlay network is used to route both query and update messages.

We denote with $\mathcal{T} = (P, E)$ a spanning tree over P, where $E \subseteq \{\{P_i, P_j\} \mid 1 \leq i, j \leq N\}$ is the set of links connecting pairs of nodes. In a system with N nodes, there are $N-1$ links on the spanning tree. For each $P_i \in P$, we denote with *Nb* (P_i) the set of neighbors of P_i, that is, the set of all peers directly connected to P_i on the overlay network.

Let $\mathcal{T}(P_i \to P_j)$ be the subtree of $\mathcal{T}$ that contains P_j and does not contain $P_i \in \mathit{Nb}(P_j)$. That is, $\mathcal{T}(P_i \to P_j)$ is the subtree containing node P_j which has been obtained after removing the link $\{P_i, P_j\}$ from $\mathcal{T}$ (see Fig. 1).

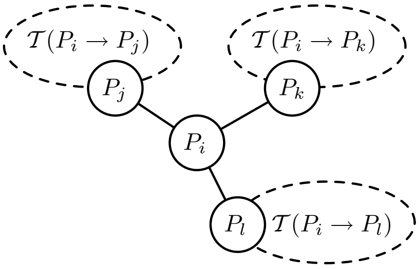

$\mathcal{T}(P_i \to P_j)$
$\mathcal{T}(P_i \to P_k)$
P_j
P_k
P_i
P_l
$\mathcal{T}(P_i \to P_l)$

Fig. 1. A portion of the tree-shaped overlay network

Each peer P maintains a summary of all the information of its local resources as follows. If the domain of attribute A is the interval $[a, b]$, we select $k + 1$ division points $a = a_0 < a_1 < \ldots < a_k = b$ such that $[a, b]$ is partitioned into k disjoint intervals $[a_i, a_{i+1})$, $i = 0, 1, \ldots k - 1$. Given an attribute A we encode the value $D[A]$ with a k bit binary vector $\mathit{BitIdx}(D[A]) = (b_0, b_1, \ldots, b_{k-1})$, such that $b_i = 1$ if and only if $D[A] \in [a_i, a_{i+1})$. Both the parameter k (number of bits of the bit vector) and the division points $a_0, a_1, \ldots a_k$ may be different for each attribute type. The final local bitmap index for an attribute A on peer P is obtained by a bitwise `OR` operation between the indices of all data items D of P:

$$\mathit{BitIdx}(P, A) \equiv \bigvee_{D \in \mathit{Data}(P)} \mathit{BitIdx}(D[A]) \tag{1}$$

where $\mathit{Data}(P)$ is the set of all the resources of P.

Let us consider a generic peer P_i. For each neighbor $P_j \in \mathit{Nb}(P_i)$, P_i keeps information on the data items which can be found by following the link $\{P_i, P_j\}$ on the overlay network. For this purpose, P_i maintains an index called $\mathit{LinkBitIdx}(P_i \to P_j, A)$ for each attribute A of each data item D in $\mathcal{T}(P_i \to P_j)$. Since P_i can receive information only from its neighborhood, the index is calculated recursively in the following way:

$$\mathit{LinkBitIdx}(P_i \to P_j, A) \equiv \mathit{BitIdx}(P_j, A) \bigvee (\bigvee_{P \in \mathit{Nb}(P_j) - P_i} \mathit{LinkBitIdx}(P_j \to P, A)) \tag{2}$$

The final result is that the index contains the bitwise union of all the bitmap indices $\mathit{BitIdx}(P, A)$ associated with every peer in $\mathcal{T}(P_i \to P_j)$. Note that $\mathit{LinkBitIdx}(P \to P', A)$ is a binary string of the same size of $\mathit{BitIdx}(D[A])$, with possibly more than one bit set to 1.

Fig. 2 shows a P2P network with a single attribute A_1, whose values are encoded with a 4-bit vector index. The binary strings in the shaded boxes represent the bit vector indices for the local repository; binary strings in the small white boxes represent the values of $\mathit{LinkBitIdx}(P \to P', A_1)$. For example, node E has a local

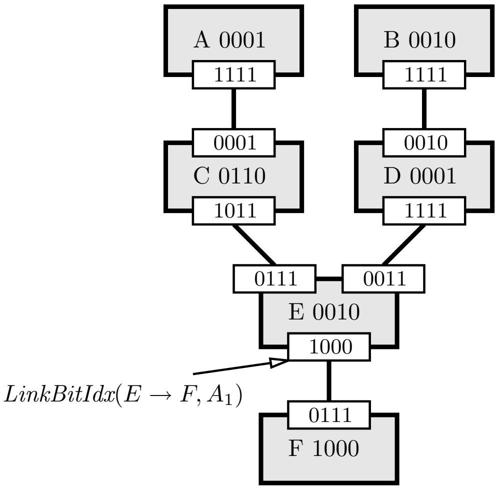


Fig. 2. Example of P2P network with bit vector indices.

data item D with *BitIdx* $(D[A_1]) = 0010$; the value of *LinkBitIdx* $(E \to F, A_1)$ is 1000.

Observe that *LinkBitIdx* $(A \to C, A_1)$ is, according to Eq. 2, the logical "or" of the bit vector representation of values of $D[A_1]$ on nodes B, C, D, E, F.

4.1 Handling Queries

We now illustrate how queries are processed. We assume that queries originate from any node P in the system. As in Gnutella Gnutella (2006), queries are propagated from node P to its neighbors using a Breadth First Search (BFS) algorithm; however, unlike Gnutella, queries are not necessarily routed to all neighbors: our system performs a Directed BFS (DBFS) over the tree overlay network. The DBFS is driven by the vector indices associated with individual peers connections.

Recall from the previous section that node P knows the bit vector *LinkBitIdx* $(P \to P', A)$, for each $P' \in$ *Nb* (P), where *LinkBitIdx* $(\cdot)$ is defined according to Eq. 2. Suppose that node P receives query $Q := v_1 \leq A \leq v_2$ from one of its neighbors P_{in}. The query is propagated along the connection from P to $P_{out} \in$ *Nb* $(P) - P_{in}$ if a match is likely to be present in $\mathcal{T}(P \to P_{out})$. A necessary condition for the existence of a match is that the logical "and" between *LinkBitIdx* $(P \to P_{out}, A)$ and the bit vector representation of the interval $[v_1, v_2]$ is nonzero.

Algorithm 1 illustrates the pseudocode executed by P to process a query message. Upon receiving a query from neighbor P_{in}, the query is forwarded to the remaining neighbors which have a potential match. Results are fanned back to P_{in}, until they eventually reach the originator. Note that this approach only works if the overlay network is guaranteed to be acyclic (i.e., is a tree), as we are assuming. The result of a query is the set of all peers with local data items matching the search criteria. For sake of readability we only present the case in which the query comes from another node in the network. In case of it originated directly from a user, P performs exactly the same operations, using an undefined value for P_{in} and returning the results back to the user.

We show in Algorithm 2 the function *Match* $(Q, P_i \to P_j)$, which is used to test for a potential match of query Q on the subtree $\mathcal{T}(P_i \to P_j)$. Query Q is decomposed according to the grammar described in the previous section. For each instance of the terminal production $Q := v_1 \leq A \leq v_2$, the function compares the bit vector representation of interval $[v_1, v_2]$ with *LinkBitIdx* $(P_i \to P_j, A)$. If the intersection is zero, then no match exists on $\mathcal{T}(P_i, P_j)$. If the intersection is nonzero, then there *may* be a match on $\mathcal{T}(P_i, P_j)$.

Algorithm 1 lookup(Q) executed by peer P

```
loop
  Wait for query $Q$ from some $P_{in} \in Nb(P)$
  Let $R := \emptyset$                                        {Query result}
  for all $P_{out} \in Nb(P) - P_{in}$ do
    if $Match(Q, P \to P_{out})$ then
      Relay $Q$ to $P_{out}$
      Let $R'$ be the reply reported by $P_{out}$
      Let $R := R \cup R'$
  if There are local matches to $Q$ then
    Let $R := R \cup$ the set of local local resources matching $Q$
  Report $R$ to $P_{in}$
```

Algorithm 2 *Match* $(Q, P_i \to P_j)$

```
if $Q = Q_1$ and $Q_2$ then
  Return $Match(Q_1, P_i \to P_j) \wedge Match(Q_2, P_i \to P_j)$
else if $Q = Q_1$ or $Q_2$ then
  Return $Match(Q_1, P_i \to P_j) \vee Match(Q_2, P_i \to P_j)$
else if $Q := v_1 \leq A \leq v_2$ then
  Let $a_0, a_1, \ldots a_k$ be the subdivision points for $A$
  for all $i = 0 \ldots k - 1$ do
    Let $b_i = \begin{cases} 1 & \text{if } [a_i, a_{i+1}) \cap [v_1, v_2] \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$
  Let $B := (b_0, b_1, \ldots b_{k-1})$
  Return $(LinkBitIdx(P_i \to P_j, A) \wedge B \neq 0)$
```

4.2 Handling Updates and Insertions

We now describe how updates can be processed.Suppose that a peer P notes a change in $D[A]$ for a data item $D \in$ *Data* (P). Let v_{new} and v_{old} be the new and old values of $D[A]$, respectively. The first action taken by P is to compute the bitmap representation of v_{new}, *BitIdx* (v_{new}). If it is equal to *BitIdx* (v_{old}), no other actions are needed. Otherwise, P computes the new *BitIdx* (P, A). Again, it may happen that the new index is the same as the old one, and then no further actions are required. In the case that the new index differs from the previous one, an update message is propagated in order to preserve the property defined by Eq. 2. Update messages consists of the name of the attribute whose value is changed, and its up-to-date bit vector representation. The updated bit vector representation for attribute A to be associated to the link $P_{out} \rightarrow P$ can be computed by P as follows:

$$\mathit{LinkBitIdx}\,(P_{out} \rightarrow P, A) = \mathit{BitIdx}\,(D[A]) \vee \left(\bigvee_{P' \in Nb(P) - P_{out}} \mathit{LinkBitIdx}\,(P \rightarrow P', A) \right) \tag{3}$$

where *BitIdx* $(D[A])$ is the bit vector representation of $D[A]$ for data item D on node P.

Algorithm 3 describes the actions executed by peer P when it notices a change in the local data store. If the bit vector representation of the new and old values are the same, nothing is done. Otherwise, an update vector index is computed and sent to each of its neighbors.

Algorithm 3 initiate_update(A, v_{new}) executed by peer P

```
Let $v_{\text{old}} := D[A]$
if $\mathit{BitIdx}(v_{\text{new}}) \neq \mathit{BitIdx}(v_{\text{old}})$ and $\mathit{BitIdx}(P, A_{\text{new}}) \neq \mathit{BitIdx}(P, A_{\text{old}})$ then
  for all $P_{out} \in \mathit{Nb}(P)$ do
    Let $B := \mathit{BitIdx}(P, A)$
    for all $P' \in \mathit{Nb}(P) - P_{out}$ do
      Let $B := B \vee \mathit{LinkBitIdx}(P \rightarrow P', A)$
    Send bit vector $B$ for $A$ to $P_{out}$
```

Each peer executes Algorithm 4 to process update messages coming from incoming connections. It is very similar to Algorithm 3: updated bit vector indices are computed according to Eq. 3 and sent to neighbors.

Algorithm 4 process_update$()$ executed by peer P

```
loop
  Wait for bit vector $B$ for $A$ from $P_{in}$
  if $B \neq \mathit{LinkBitIdx}(P \rightarrow P_{in}, A)$ then
    Let $\mathit{LinkBitIdx}(P \rightarrow P_{in}, A) := B$
    if $\mathit{BitIdx}(P, A) \vee B \neq B$ then
      for all $P_{out} \in \mathit{Nb}(P) - P_{in}$ do
        Let $B' := \mathit{BitIdx}(P, A)$
        for all $P' \in \mathit{Nb}(P) - P_{out}$ do
          Let $B' := B' \vee \mathit{LinkBitIdx}(P \rightarrow P', A)$
        Send $B'$ to $P_{out}$
```

Insertions of new data items into the P2P system can be done with the same algorithms just described for updates. When a new data item D is registered at peer P, then for each $A \in \mathit{AttList}\,(D)$, P executes the procedure **initiate_update**$(A, D[A])$ (outgoing messages can be batched together for efficiency).

4.3 Nodes Joining and Leaving the system

In order to limit the number of hops of the messages processed in the system, it is necessary to build an appropriate overlay network on the top of the set of peers $\mathcal{P} = \{P_1, P_2, \ldots P_N\}$. The algorithms presented above rely on a tree-structured overlay network $\mathcal{T}$, which is a spanning tree over the set of nodes $\mathcal{P}$. Algorithms 1–4 are of course totally independent of the way the overlay network topology is maintained: every algorithm for maintaining a distributed spanning tree over the set of peers can be applied when nodes join or leave the network.

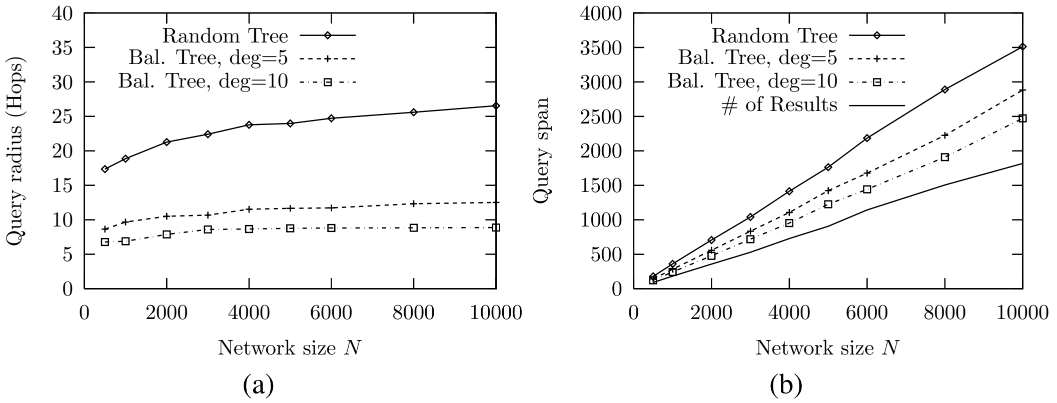
However, the performance of the system depends on the topological characteristics of the overlay network, as we will see in more details in the next section. In order to avoid degenerate cases, the overlay network should have low diameter, and such property should be maintained as nodes join and leave the system. For this purpose, it is possible to use the algorithm described in Pandurangan et al. (2003) to maintain the spanning tree $\mathcal{T}$ with bounded degree and logarithmic diameter.

4.4 Simulation results

We conducted several simulation experiments in order to evaluate the performances of this first proposal of a P2P resource discovery system. In this section we discuss the results of the simulation study.

The experimental settings are as follow. We consider an N node P2P system with single attribute data items. Attribute values are uniformly and randomly distributed in the $[0, 1]$ interval. We consider the following overlay tree network topologies: random, balanced with degree 5, and balanced with degree 10. Simulation results are computed as confidence interval with 90$\%$ confidence level. Each measurement was repeated several times in order to get confidence intervals having width of less than 5$\%$ the central value (in the figures we only show the central value).

We first analyze the maximum number of routing hops (*query radius*) needed to locate a data item as a function of network size. Fig. 3(a) shows the results for three different overlay network topologies. In Fig. 3(b) we plot the total number of queried nodes (*query span*) as a function of the network size, for different topologies. Both the query radius and query span are Lower is Better (LB) metric. The data points were calculated by performing 100 random range queries on the network, each one originating from a uniformly chosen node.



(a) (b)

Fig. 3. (a) Query radius and (b) query span as a function of the network size ($k = 32$, lower is better)

As we can see, the query radius grows as $O(\log(N))$, while the query span grows as $O(N)$, N being the size of the network. Note also in Fig. 3(b) that the number of matches is linear with the size of the network. As the query mechanism is guaranteed to locate every existing match, the number of matches is a lower bound for the query span. Thus the query span is optimal considering the number of matches.

We can also define the precision of the query routing strategy. The *query precision* is defined as the ratio between the number of data items matching the query and the number of items matching the bit vector representation of the query (*Number of real matches*/*Number of potential matches*).

In our tests, we considered a network of $N = 1000$ nodes, and performed 100 range queries given a match probability p, which represents the probability that a node matches the query. The number of data items, distributed over the N nodes of the network, was exactly equal to N. The data items were characterized by a single attribute A, with values uniformly distributed in $[0, 1]$. In addition, the $[0, 1]$ interval of the attribute values was partitioned into k equally sized bins.

The single-attribute range queries were of the form of $(v \leq A)$ **and** $(A \leq v+p)$, for v uniformly chosen in $[0, 1-p]$. Basically, we thus performed queries characterized by different *interval widths* p. However, since the attribute values were uniformly distributed in $[0, 1]$, this corresponds to a match probability exactly equal to p, as stated above in our testing hypotheses.

In Fig. 4 we show the precision of our algorithm as a function of match probability p, where the simulation setting was the one discussed above.

From the figure we see that the precision turns out to be higher as the number k of bits in the vector indices increases. Also, the precision increases for large values of the match probability p. We try to explain this behavior by deriving an analytic formulation of the query precision. Note that the expected number of data items

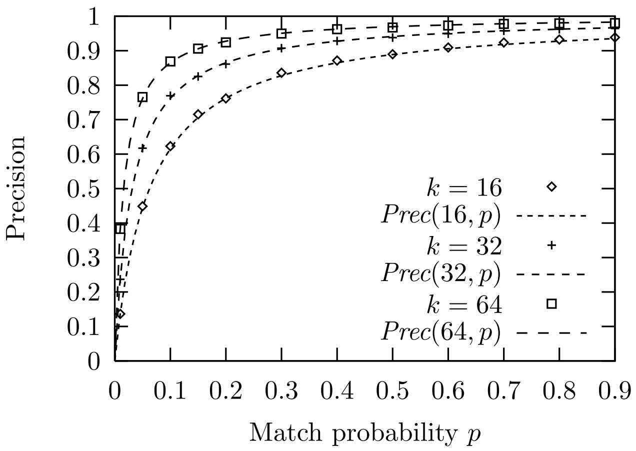


Fig. 4. Precision as a function of match probability ($N = 1000$, random tree, higher is better). Function *Prec*(k, p) is defined in 4.

matching a range query of match probability p is Np. Moreover, due to the uniform distribution of the values of attribute A in the interval $[0, 1]$, for $0 < p \leq 1 - 1/k$, the expected number of *false positives* (i.e., data items whose bit vector indices match the query, but their exact attribute values do not) is N/k. The reason is that, on average, each side of the $[v, v + p]$ interval will cover half a bin. The precision is thus $Np/(Np + N/k) = kp/(kp + 1)$. Finally, if $p > 1 - 1/k$, the expected number of false positives is $N(1 - p)$, which is the expected number of data items falling *outside* the interval $[v, v + p]$. In this case the precision is simply p. Let *Prec*(k, p) be . We can thus define *Prec*(k, p), i.e., the precision of queries with match probability p on a system with k bit indices, as follows:

$$\mathit{Prec}(k, p) = \begin{cases} \dfrac{kp}{kp + 1} & \text{if } 0 < p \leq 1 - 1/k \\ p & \text{if } 1 - 1/k < p \leq 1 \end{cases} \tag{4}$$

Fig. 4 confirms that this analytic formulation of precision is highly accurate.

We finally analyzed the behavior of the update mechanism. In Fig. 5(a) we plot the mean number of hops traversed by an update message (update radius) as a function of the network size; in Fig. 5(b) we plot the number of nodes reached by an update message (update span) as a function of the network size. From the figures we can observe that both the update radius and update span are independent of the network size. On the other hand, they are influenced by the degree of peers on the overlay network: a balanced tree of degree 10 produces larger update radius and spans than the balanced tree of degree 5, with the random overlay network topology laying in between.

Fig. 6 plots the update span as a function of the resource density δ. The resource density δ is the probability that a node contains a resource. As usual, resource attribute values are uniformly distributed in the $[0, 1]$ interval, hence the total number

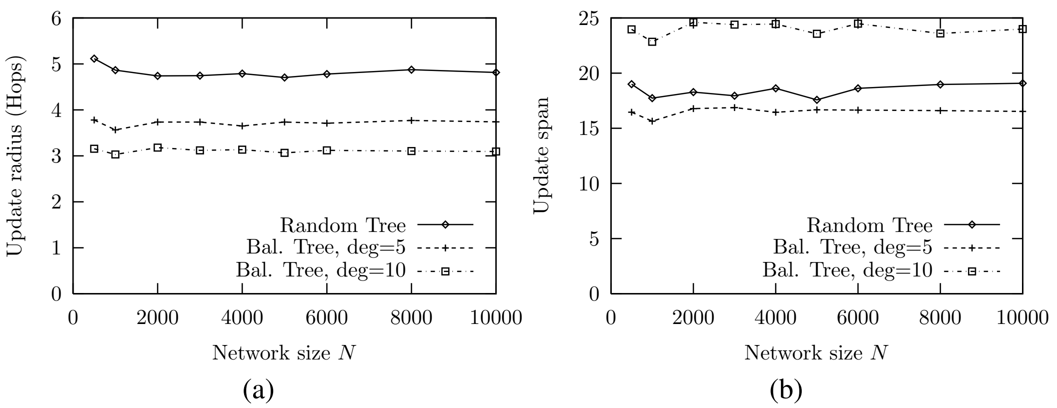


Fig. 5. (a) Update radius and (b) update span as a function of network size ($k = 16, p = 0.5$, lower is better).

of resources in an N nodes network is $N\delta$. As expected, the update span decreases for larger values of δ: high data density implies that the vector indices associated with the links have a higher density of bits set to 1, thus updates are more likely not to propagate.

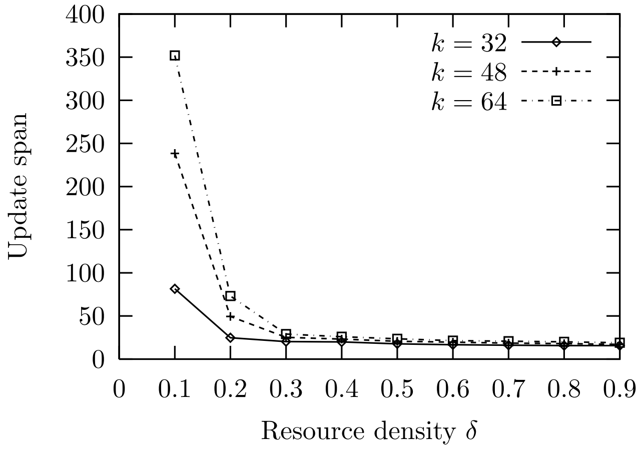


Fig. 6. Number of nodes updated as a function of δ ($N = 1000$, random topology, lower is better)

5 Forest of Trees: a P2P system exploiting a hierarchical network

Despite its good performance properties, the structure described in the previous section has two main drawbacks. First, the topology may become hard to maintain when nodes join and leave the network, especially in the case we want the tree to remain (almost) balanced. Second, the nodes close to the tree root may become overloaded of routing requests. For these reasons, in this paper we propose to partition the network into a set of small trees, i.e. a forest. Each peer P belongs to a single tree, which we call *group* of P. The nodes that have a link with P are of two types: (1) nodes of same group, i.e. *local neighbors*, denoted with $\mathit{LNb}\left(P\right)$; (2) nodes that belong to other groups, i.e. *external neighbors*, denoted with $\mathit{ENb}\left(P\right)$. The set of all the peers connected to P is called the *neighborhood* of P and is denoted with $\mathit{Nb}\left(P\right)$, i.e. $\mathit{Nb}\left(P\right) = \mathit{LNb}\left(P\right) \cup \mathit{ENb}\left(P\right)$. In order to have a connected network we impose that for each group of nodes, at least a node of the group must have an external neighbor, i.e. $\forall$ group $G \; \exists \; P \in G \mid \mathit{ENb}\left(P\right) \neq \emptyset$. An example of a network of this type is shown in Fig. 7. For the sake of simplicity, the figure presents a *vertex clustering* of the network, i.e. only connections between groups are drawn.

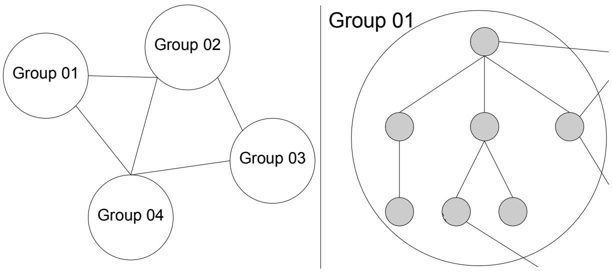


Fig. 7. An example of the new network organization

We now show how routing information and update schemes change with the new network structure, by first discussing the new bitwise RIs.

5.1 Bitmap Indices

The information associated with of the local resources of a peer is computed in the same way as for the previous network.

On the other hand, since we now have internal and external neighbors, we have to change the definition of the information associated with each outgoing link from a node. More precisely, given a peer P we want to make a distinction between the information related to resources present in the same group of P, and that related to the resources that P can reach through the external neighbors of either P or all the other nodes belonging to the group of P.

Let us change the definition of Eq. 2 in the following way:

$$\mathit{LinkBitIdx}\left(P_i \rightarrow P_j, A\right) \equiv \mathit{BitIdx}\left(P_j, A\right) \bigvee \Big(\bigvee_{P \in \mathit{LNb}(P_j) - P_i} \mathit{LinkBitIdx}\left(P_j \rightarrow P, A\right)\Big) \tag{5}$$

As can be seen, the $\mathit{LinkBitIdx}\left(\cdot\right)$ quantity is now computed over only the local neighbors of a node. Thus, each peer P_i computes the quantity $\mathit{LinkBitIdx}\left(P_i \rightarrow P_j, A\right) \forall P_j \in \mathit{LNb}\left(P_i\right)$. This is used as an *internal routing index*, since it describes only the resources available inside a given group.

As said in the previous section, each group has at least one external neighbor, i.e., there exists at least one of its member nodes that has an external connection with a node in another group. We want to exploit this fact and spread external information within the group, in order to allow each peer to reach resources that are located in groups different from its own. Let G be a group. We want each peer of G to know not only the resources it can find inside G, but also the resources that are reachable through external connections of the nodes of G. Let P_i be a node of G. P_i can directly reach resources that are located in an external group G_{ext}, connected to G, if P_i has at least one node of G_{ext} among its external neighbors. Otherwise, it can reach them by routing a query to another internal node of G that has a connection with G_{ext}. To this end, P_i maintains an *external bitmap index* that is computed in the following way.

$$\mathit{ExtBitIdx}\left(P_i \to P_j, A\right) = \begin{cases} \mathit{LinkBitIdx}\left(P_i \to P_j, A\right) & \text{if } P_j \in \mathit{ENb}\left(P_i\right) \\ \displaystyle\bigvee_{P \in \mathit{Nb}(P_j) - P_i} \mathit{ExtBitIdx}\left(P_j \to P, A\right) & \text{if } P_j \in \mathit{LNb}\left(P_i\right) \end{cases} \tag{6}$$

where $\mathit{Nb}\left(P_j\right) = \mathit{ENb}\left(P_j\right) \cup \mathit{LNb}\left(P_j\right)$.

In the equation above, $\mathit{LinkBitIdx}\left(P_i \to P_j, A\right)$ is computed as for Eq. 5. It is worth noting that in this case, since $P_i \notin \mathit{LNb}\left(P_j\right)$, $\mathit{LinkBitIdx}\left(P_i \to P_j, A\right)$ gives to P_i a complete summary of all the resources that can be found in the nodes of group G_{ext}, where $P_j \in G_{ext}$.

More generally, if $P_j \in \mathit{ENb}\left(P_i\right)$, then the new index $\mathit{ExtBitIdx}\left(P_i \to P_j, A\right)$ gives to P_i a description of the resources of an external group G_{ext}, where $P_j \in G_{ext}$. Conversely, if $P_j \in \mathit{LNb}\left(P_i\right)$, then $\mathit{ExtBitIdx}\left(P_i \to P_j, A\right)$ gives to P_i a description of the external resources (belonging to neighboring groups) that are available through external connections of the internal nodes that belong to the subtree rooted on the local neighbor P_j. Fig. 8 shows an example of a system with two groups G_1 and G_2; only external indices of nodes A and C are shown. Internal indices are omitted, as they are computed in exactly the same way as the single tree case.

5.2 Query Routing

The routing phase varies too. As for the previous case, when a peer P receives a query $Q := v_1 \leq A \leq v_2$ from a neighbor P_{in}, it forwards it to neighbor $P_{out} \in \mathit{Nb}\left(P\right) - P_{in}$ only if a match is likely to be present in $\mathcal{T}(P \to P_{out})$ or in at least one of the external groups connected to the nodes in $\mathcal{T}_G(P \to P_{out})$. To this end, the result of the logical AND between the bitmap representation of the query range, and at least one of both $\mathit{LinkBitIdx}\left(P \to P_{out}, A\right)$ and $\mathit{ExtBitIdx}\left(P \to P_{out}, A\right)$ must be a non-zero vector.

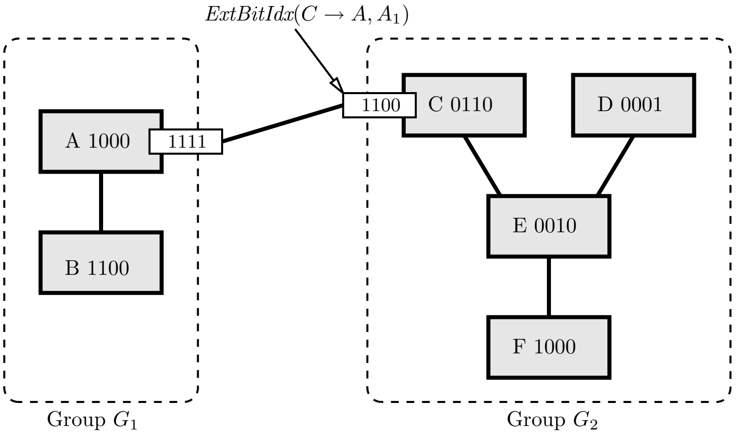
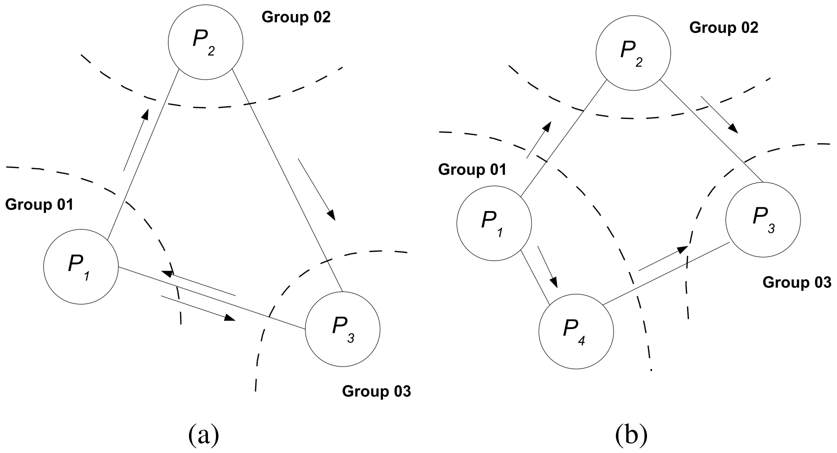


Fig. 8. Example of group indices

As a consequence of the creation of two types of indices, we also have two possible types of matches. A query is forwarded to a node if the at least one between the internal and external indices shows a match.

The routing scheme described so far may not be enough for an efficient routing of queries. Note that the outer graph, associated with the inter-group connections, is now unstructured. Therefore, it may also include loops among the (super) nodes, each corresponding to a distinct group of nodes G. Let us consider the situations depicted in Fig. 9(a). For the sake of simplicity, only links between the presented nodes are shown. In the first case, P_1, P_2 and P_3 have at least a resource matching query Q. Let us suppose the P_1 receives Q. After P_1 has detected the local match, it forwards the query to its neighbors. Node P_2 forwards the query to P_3. P_3 will then send the query back to P_1, that, in turn, will forward it again to P_2. The final result is an infinite loop. In order to avoid loops, we need to improve the routing scheme.



(a)

(b)

Fig. 9. Examples of loops (a) and multiple resolutions of the same query (b)

We add a further information to that associated with a query message: the list of all the IDs that are associated with the node groups that were already traversed by the query. Let *GIDs*$_Q$ be such a list. This is useful to prevent forwarding multiple times the same query Q from a group to another, when more than a single links connect nodes of the groups. When a peer P forwards Q to P', where P and P' belong to distinct groups, P adds the group ID of P' to $GIDs_Q$. In this way, the nodes belonging to the same group of P forward Q to an external neighbor P' only if the group ID of P' is not listed in $GIDs_Q$. Nothing changes for the internal neighbors. This solution eliminates cycles, but does not completely avoid the problem illustrated in Fig. 9(b). A query Q is forwarded by node P_1 to all nodes P_2, P_3 and P_4 that match it. As can be seen, node P_3 receives the same query from node P_2 and node P_4, which belong to distinct groups. The previously described mechanism does not prevent P_3 from processing the same query twice. For this reason we need to associate with each peer P a *query cache*, i.e. a cache that contains the latest *m* query IDs received by P. If P receives a query whose ID is already listed in its own query cache, it simply discards the query, avoiding processing it twice.

The new query routing method is described in Algorithms 5 and 6. As for the corresponding algorithm for the single tree, in Algorithm 5 we only show the case in which the query comes from another node in the network. The differences between the two cases (query coming from a neighbor or originated from the user) consist only in the fact that in the latter case P adds its own group ID to the groups traversed by query Q, poses an undefined value for P_{in} and return the results back to the user.

Algorithm 5 lookup(Q) executed by peer P

```
loop
  Wait for query $Q$ from some $P_{in} \in Nb(P)$
  if $Q \notin P.QueryCache$ then
    Add $Q$ to $P.QueryCache$
    Let $R := \emptyset$                                    {Query result}
    for all $P_{out} \in Nb(P) - P_{in}$ do
      if $P_{out} \in ENb(P)$ then
        if $P_{out}.Group \notin Q.Groups$ then
          Add $P_{out}.Group$ to $Q.Groups$
        else if $P_{out}.Group \notin Q.Groups$ then
          Continue
      if $Match(Q, P \rightarrow P_{out})$ then
        Relay $Q$ to $P_{out}$
        Let $R'$ be the reply reported by $P_{out}$
        Let $R := R \cup R'$
    if There are local matches to $Q$ then
      Let $R := R \cup \{P\}$
    Report $R$ to $P_{in}$
```

Algorithm 6 *Match* $(Q, P_i \to P_j)$

```
if $Q = Q_1$ and $Q_2$ then
  Return Match$(Q_1, P_i \to P_j) \wedge$ Match$(Q_2, P_i \to P_j)$
else if $Q = Q_1$ or $Q_2$ then
  Return Match$(Q_1, P_i \to P_j) \vee$ Match$(Q_2, P_i \to P_j)$
else if $Q = v_1 \le A \le v_2$ then
  Let $a_0, a_1, \dots a_k$ be the subdivision points for $A$
  for all $i = 0 \dots k-1$ do
    Let $b_i = \begin{cases} 1 & \text{if } [a_i, a_{i+1}) \cap [v_1, v_2] \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$
  Let $B := (b_0, b_1, \dots b_{k-1})$
  Return (LinkBitIdx$(P_i \to P_j, A) \wedge B \neq 0)$ or (ExtBitIdx$(P_i \to P_j, A) \wedge B \neq 0)$
```

5.3 Handling Updates

The new network topology involves changes for the update scheme too. In particular, when a local update happens, the new internal bit vector is computed according to Eq. 5. Similarly, if there is a change in an external node, the *ExtBitIdx* $(\cdot)$ index associated with the inter-group link is updated. Then the change is propagated inside the group to only update the internal *ExtBitIdx* $(\cdot)$ indices, according to Eq. 6.

In other words, if G' is the originating group, i.e., the group which the node that originated the update belongs to, this update is first propagated to the nodes of G', and then, if the global index that represent G' is changed, to external nodes belonging to groups connected to G'. The goal of the last propagation is only to update the *ExtBitIdx* $(\cdot)$.

Note that, as for the single tree case, the propagation of an update is stopped when it reaches a node where the new and old indices do not differ. Keeping this remark in mind, note that if a change happens on the local resources of a node P_i of a group G and this changes the RI for G, it has to be communicated to the external neighbors of G, in order to respect Eq. 6, thus transforming a local event into an external communication. An update can then be transmitted both internally, as an update of *LinkBitIdx* $(\cdot)$, and externally, as an update of *ExtBitIdx* $()$. When it reaches an external group, it is forwarded inside that group only as an update of the *ExtBitIdx* $()$ index. It is worth remarking that update messages directed to an external group are not further propagated outside that group.

Algorithms 7 and 8 report the pseudo-code used to perform updates in the network.

Algorithm 7 initiate_update(A, v_{new}) executed by peer P

```
Let $v_{\text{old}} := D[A]$
if $\mathit{BitIdx}(v_{\text{new}}) \neq \mathit{BitIdx}(v_{\text{old}})$ and $\mathit{BitIdx}(P, A_{\text{new}}) \neq \mathit{BitIdx}(P, A_{\text{old}})$ then
  for all $P_{out} \in \mathit{Nb}(P)$ do
    Let $B := \mathit{BitIdx}(P, A)$
    for all $P' \in \mathit{Nb}(P) - P_{out}$ do
      Let $B := B \vee \mathit{LinkBitIdx}(P \to P', A)$
    Send bit vector $B$ for $A$ to $P_{out}$
```

Algorithm 8 process_update$()$ executed by peer P

```
loop
  Wait for bit vector $B$ for $A$ from $P_{in}$
  if $P_{in} \in \mathit{LNb}(P)$ then
    if $B$ is a $\mathit{LinkBitIdx}(\cdot)$ and $B \neq \mathit{LinkBitIdx}(P \to P_{in}, A)$ then
      Let $\mathit{LinkBitIdx}(P \to P_{in}, A) := B$
      for all $P_{out} \in \mathit{Nb}(P) - P_{in}$ do
        Let $B' := \mathit{BitIdx}(P, A)$
        for all $P' \in \mathit{LNb}(P) - P_{out}$ do
          Let $B' := B' \vee \mathit{LinkBitIdx}(P \to P', A)$
        Send $B'$ to $P_{out}$
    else if $B$ is a $\mathit{ExtBitIdx}(\cdot)$ and $B \neq \mathit{ExtBitIdx}(P \to P_{in}, A)$ then
      Let $\mathit{ExtBitIdx}(P \to P_{in}, A) := B$
      for all $P_{out} \in \mathit{LNb}(P) - P_{in}$ do
        Let $B' := (0, 0, \ldots 0)$
        for all $P' \in \mathit{Nb}(P) - P_{out}$ do
          Let $B' := B' \vee \mathit{ExtBitIdx}(P \to P', A)$
        Send $B'$ to $P_{out}$
  else if $P_{in} \in \mathit{ENb}(P)$ then
    if $B \neq \mathit{ExtBitIdx}(P \to P_{in}, A)$ then
      Let $\mathit{ExtBitIdx}(P \to P_{in}, A) := B$
      for all $P_{out} \in \mathit{LNb}(P) - P_{in}$ do
        Let $B' := (0, 0, \ldots 0)$
        for all $P' \in \mathit{Nb}(P) - P_{out}$ do
          Let $B' := B' \vee \mathit{ExtBitIdx}(P \to P', A)$
        Send $B'$ to $P_{out}$
```

5.4 Experimental results

In this section we report the results of many simulation experiments performed to study how different measures–like query radius (hops), query span, precision, recall, etc.–change as a function of various parameters and network configuration.

It is worth noting that, in a P2P system based on Forest of Tree, a query can be stopped during its propagation, thus not reaching all the nodes matching a given query. So, it becomes important to also study the *recall* of a query, i.e., the ratio between the number of (real) matches returned by a query, and the total number of matches existing in the whole network.

We start this discussion by an analytic evaluation of query radius and update propagation.

Analytical evaluation: Query radius

In this section we give an analytical expression for the mean number of groups visited by a query; this quantity is directly related to the *query radius*, that is, the maximum number of hops a query traverses.

Let us consider a chain of $L + 1$ groups $G_1, G_2, \ldots G_{L+1}$, where for each i, $1 < i < L + 1$, group G_i is connected with groups G_{i-1} and G_{i+1}. We assume that a query originates in a node of group G_1. We denote with p the match probability, and with N_G the mean number of nodes in each group. The probability P_{Gnomatch} that no match is found in a group can be expressed as:

$$P_{\text{Gnomatch}} = (1 - p)^{N_G}$$

Thus, $P_{\text{Gmatch}} = 1 - P_{\text{Gnomatch}}$ is the probability that at least one value matching the query can be found in the group. According to the query routing algorithm, inter-group query propagation stops when the query reaches a group whose neighbors (apart the one from which the query originated) do not have matches. We denote with X_L the discrete random variable representing the *number of inter-group hops* of a query over a chain of $L + 1$ groups (thus, $X_L \in 1, 2, \ldots L$). We know that $X_L = 0$ when G_2 contains no matches (the originator group may or may not contain matches). This happens with probability P_{Gnomatch}. In general, for every integer $i \in 0, 1, \ldots L$, the probability $\text{Pr}\,(X_L = i)$ that a query of selectivity s is passed through i hops can be expressed as:

$$\text{Pr}\,(X_L = i) = \begin{cases} P_{\text{Gnomatch}} P_{\text{Gmatch}}^{i} & \text{if } i < L \\ P_{\text{Gmatch}}^{L} & \text{if } i = L \end{cases}$$

The mean value of X_L is:

$$E[X_L] = \sum_{i=0}^{L} i\,\text{Pr}\,(X_L = i) = \frac{P_{\text{Gmatch}}(P_{\text{Gmatch}}^{L} - 1)}{P_{\text{Gmatch}} - 1}$$

We show in Fig. 10 the value of $E[X_L]$ as the chain length increases, for different values of query selectivity s and mean group population N_G; the corresponding value of P_{Gmatch} is shown in Table 1. The graphs show the horizontal limit $P_{\text{Gmatch}}/(1 - P_{\text{Gmatch}})$: this means that as the chain length increases, the expected number of visited groups is bounded by the quantity $P_{\text{Gmatch}}/(1-P_{\text{Gmatch}})$. In terms of our algorithm, this means that long chains of groups tend to degrade the *recall*, as the query routing algorithm is likely to stop forwarding the query early, even if matches could be found later on the chain. It is therefore very important to build the inter-group links by trying to shorten the group chain lengths, , i.e., to build networks where the average shortest paths, computed over the inter-group links, are kept short.

Mean group size N_G	**Match probability** p	P_{Gmatch}
50	0.01	0.395
100	0.02	0.636
200	0.01	0.866

Table 1
Parameter combinations used in Fig. 10

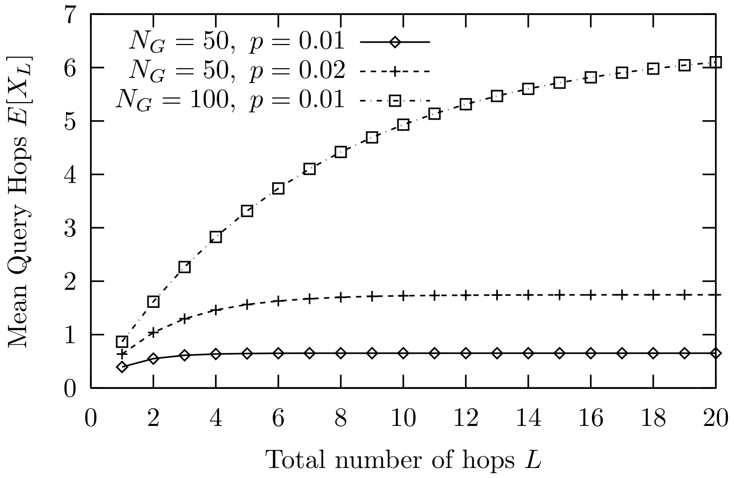


Fig. 10. Plot of the average number of inter-group hops $E[X_L]$ as a function of the total number of hops L.

Analytical Evaluation: Update Propagation

We now analyze the propagation of updates in the network. In section 5.1 we described the two types of indices used in the system. The *LinkBitIdx* $(\cdot)$ is maintained by each peer P as an *internal* RI. Then, it describes only the locally available resources, i.e., those available inside P's group. Since the internal topology of a group is a tree, the updates concerning the *LinkBitIdx* $(\cdot)$ are handled in the same way as the updates for the single tree network. Thus, the update span and radius inside a group will be the same of a single tree with a number of nodes equal to the group size.

The main difference between the two models is then determined by the presence of the *ExtBitIdx* $(\cdot)$ index. As said in section 5.3, if a node P updates one of its local attribute index, it has to communicate such a change to its external neighbors, but only if the global description of the P's group associated with that attribute changes as a consequence of the update. This fact can be derived also by the formulation of the *ExtBitIdx* $(\cdot)$ index given by Eq. 6.

The most expensive case in terms of communications occurs when the internal diffusion of an update generated by P also leads to a diffusion of an *ExtBitIdx* $()$ update to all the external neighbors of the P's group. However, it is worth recalling that when this update reaches an external group, it is no longer forwarded outside that group. Then the maximum number of nodes contacted is represented by the sizes of all the neighboring groups of the P's group.

As previously stated, this worst-case scenario happens only if the P local update on an attribute A implies a change in the group's bitmap index of A . Suppose that attribute A is indexed by a k-bit bitmap index, and that the change on P implies a change in the i-th position of the index. Let G_P be the group of P. The G_P representation for attribute A will not change if and only if $\exists P_j \in G_P, P_j \neq P$, such that it has a resource whose value for A is indexed exactly in the i-th position of the index.

Let p_i be the probability that a node has a resource whose index for A is set in the i-th position. If the attribute values are uniformly distributed over its domain, then $p_i = 1/k$. Then, $(1 - p_i)$ represents the probability for a node to not have such a setting in the index for attribute A. Let $n = |G_P|$. If an update on node P changes the $i - th$ position of the index, the probability that the global G_P index is also changed is: $(1 - p_i)^{n-1}$.

Thus, the larger a group size, the lower are the chances that a local change must be forwarded to external neighbors. Then, when groups are large enough, updates will mainly involve internal nodes. So the update span and radius will tend to be similar to the ones obtained for a single tree network.

Simulation results

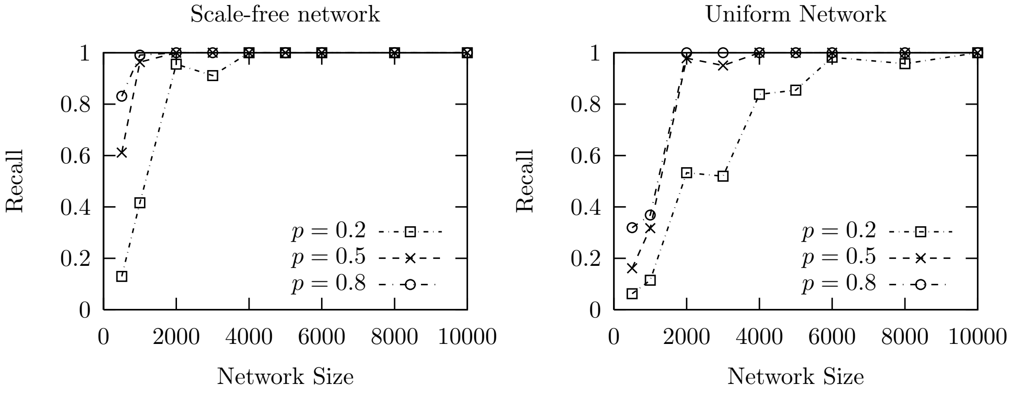
The performances of the systems has been evaluated by simulation. Simulation results were computed as confidence interval with 90% confidence level and each measurement was repeated multiple times in order to get confidence intervals having width of less than 5% of the central value.

We have previously stated that, while the groups inner structure remains a tree, the inter-group connections need not to be structured. In our simulation settings we considered two different group organizations. The first one is a scale-free network (Barabási and Reka, 1999), while the second one is a network of groups where each neighbor is uniformly chosen between all the groups, and the degrees of groups are uniformly distributed with a small variance. For both network organizations, we used the same average *group degree*, i.e., the mean number of neighbors. Moreover, also the number of edges connecting single nodes are almost the same in both networks.

In order to analyze the behavior of the system with respect to the group organization, we evaluated the performance of the system as a function of the network size, for each of the two above mentioned network topologies. We set the mean group size (number of nodes per group) as a fraction of the total number of peers in the system. This fact implies that the number of groups remains the same, while the number of nodes inside each group grows linearly with the network size. In our tests, if n is the mean group size, the actual number of nodes per groups varies uniformly in the interval $[\frac{1}{3}n, \frac{5}{3}n]$.

We will now present the simulation results for both network topology with respect to multi-attribute query resolution, propagation and update diffusion. The following results were obtained by using three different average values n of the group size: 0.75%, 1.5%, and 3% of the total number of nodes (network size).

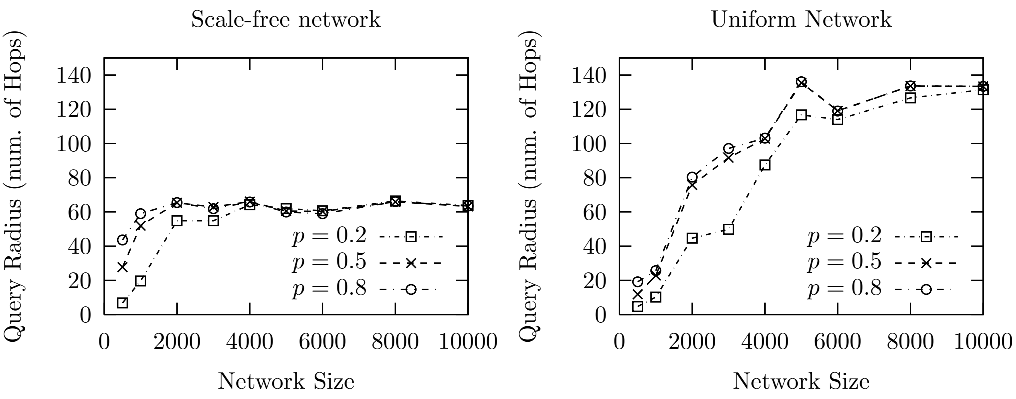
In order to evaluate the query resolution performance, we performed 100 different random queries for each network size, each one originating from a uniformly chosen node. As results, we show the average recall and the query radius (i.e., the maximum number of hops the query performed during diffusion) of these queries. See Figures 11 through 16, which show these measures for both group topologies and different group sizes. We considered three different values for p, the probability that a node matches a query. More precisely, we considered $p \in \{0.2, 0.5, 0.8\}$.



(a) (b)

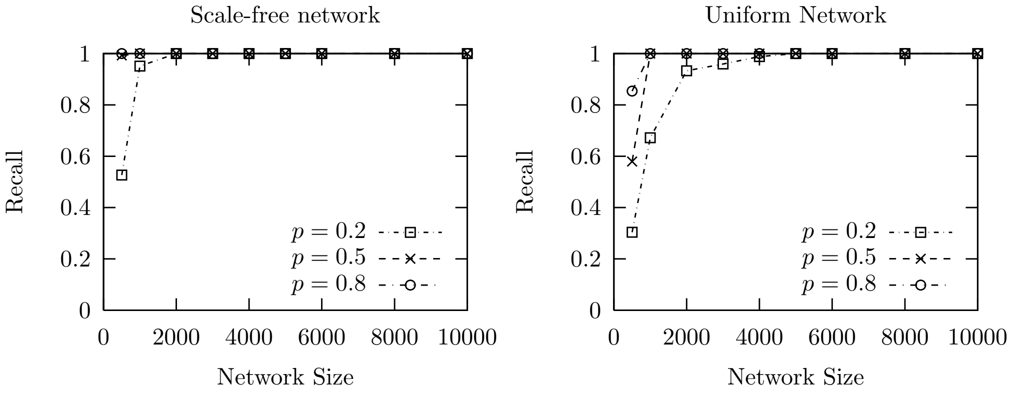
Fig. 11. Recall for the scale-free(a) and the uniform distributed (b) networks, for $n = 0.75\%$

Looking at Figures 11, 13 and 15, we observe two facts about recall: (1) the recall is generally better for the scale-free network; (2) the recall is better for larger network sizes. The first finding is due to the fact that the diameter and the average shortest path of scale-free networks are lower than that of the uniform networks. Thus, in the latter network topology, the query has to span more groups in order to reach all the nodes that have potential matches. Let G_Q be the group where the query originated and let G_i be another group in the network. The longer the path between



(a) (b)

Fig. 12. Query radius for the scale-free(a) and the uniform distributed (b) networks, for $n = 0.75\%$



(a) (b)

Fig. 13. Recall for the scale-free(a) and the uniform distributed (b) networks, for $n = 1.5\%$

G_Q and G_i, the higher the probability that a group with no matches lies in this path, as explained in Section 5.4. This fact implies that the forwarding along that path will likely be stopped, as it will not possible to route a query through a group with no matches (see Section 5.2). Obviously, the probability that a group has a match decreases, when either its size or the node match probability p decreases. This explains the behavior pointed out in the second observation above.

In Table 2 we use the definitions of Section 5.4 to compute P_{Gmatch} for each group in the network with respect to the network size, single node matching probability p and the group size n, expressed as a ratio of the network dimension.

A value $P_{Gmatch} = 1$ in this table means that $1-\epsilon \leq P_{Gmatch} \leq 1$, where $\epsilon = 10^{-12}$.

The radius of queries are shown in figures 12, 14 and 16. The radius are obviously lower when the queries are not able to reach all the nodes. In these cases, the value of p becomes relevant, as it also determines the matching probability of each group. As stated previously, the fact that a group does not contain any matching node may

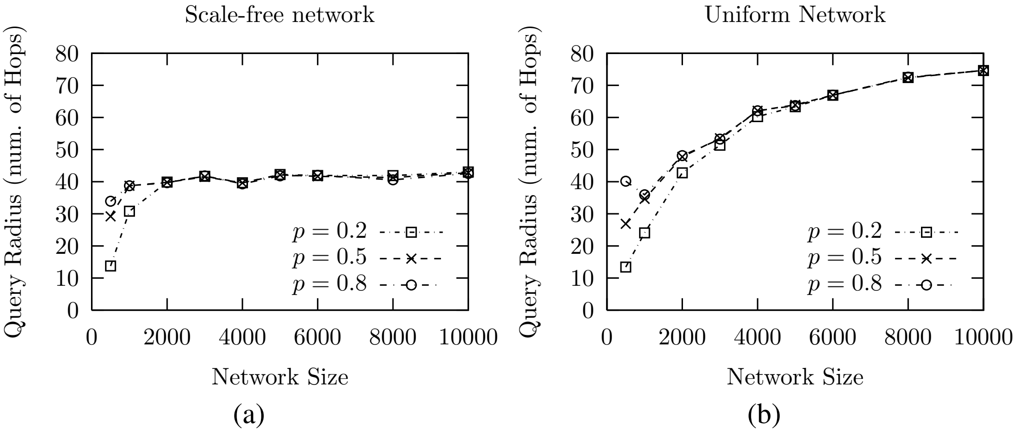


Fig. 14. Query radius for the scale-free(a) and the uniform distributed (b) networks, for $n = 1.5\%$

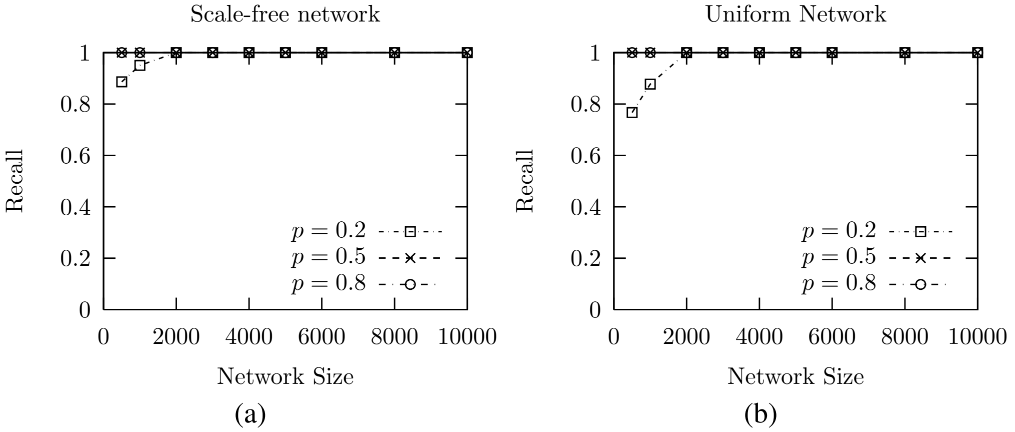
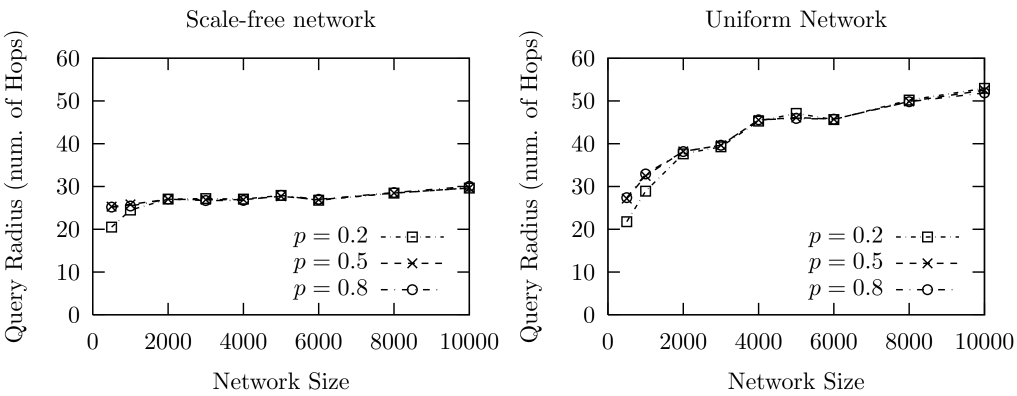


Fig. 15. Recall for the scale-free(a) and the uniform distributed (b) networks, for $n = 3.0\%$

break the routing chain from the originator node to the other peers and then its radius. The query radius is smaller in the scale-free network, since its diameter is also smaller. Moreover, when p is high enough to allow each matching node to receive the query, the query radius seems to stabilize in the scale-free case. In the uniform network it grows as the size of the system increases. The diameter of a scale-free network is not only lower than that of the other type of network, but it is also less influenced by the growth of the network.

The settings of the simulations above led to networks with only tens of groups. The largest number of groups is obtained in the first case ($n = 0.75$), corresponding to less than 150 groups. To study the network behaviour with a larger number of groups we used a lower value for n. In particular, in the simulations shown in Figures 17 and 18, we used $n = 0.2$. This means that the network has over than 500 groups. Another implication of this choice is that we had to test the system with larger sizes of the network, in order to have a reasonable number of nodes per group. So, the network size ranged from 5,000 to 100,000 nodes. These results confirm the observations of the previous cases.



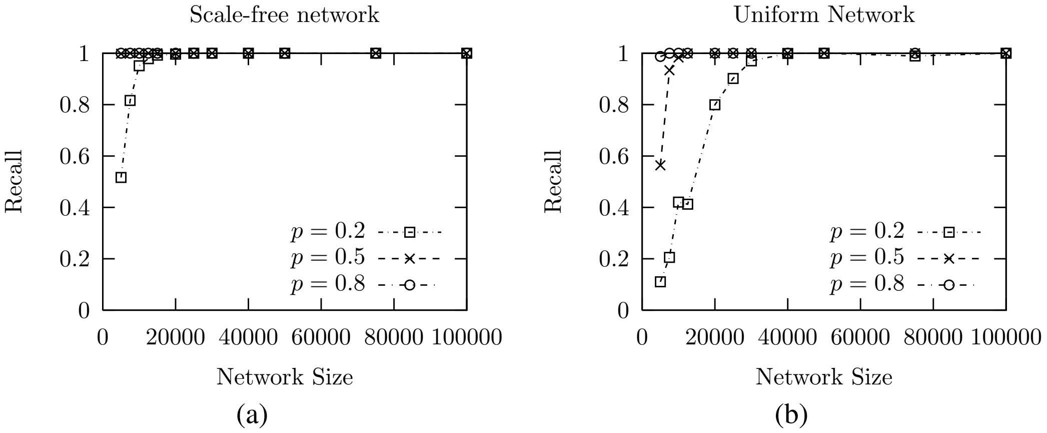
(a) (b)

Fig. 16. Query radius for the scale-free(a) and the uniform distributed (b) networks, for $n = 3.0\%$

Net Size	$p = 0.2$			$p = 0.5$			$p = 0.8$		
	$n = 0.75$	$n = 1.5$	$n = 3.0$	$n = 0.75$	$n = 1.5$	$n = 3.0$	$n = 0.75$	$n = 1.5$	$n = 3.0$
500	0.5904	0.9375	0.9984	0.8322	0.9961	0.9999	0.9648	0.9999	0.9999
1000	0.8322	0.9961	0.9999	0.9648	0.9999	0.9999	0.9988	0.9999	1
2000	0.9648	0.9999	0.9999	0.9988	0.9999	1	0.9999	1	1
3000	0.9941	0.9999	1	0.9999	1	1	0.9999	1	1
4000	0.9988	0.9999	1	0.9999	1	1	0.9999	1	1
5000	0.9998	0.9999	1	0.9999	1	1	1	1	1
6000	0.9999	1	1	0.9999	1	1	1	1	1
8000	0.9999	1	1	0.9999	1	1	1	1	1
10000	0.9999	1	1	1	1	1	1	1	1

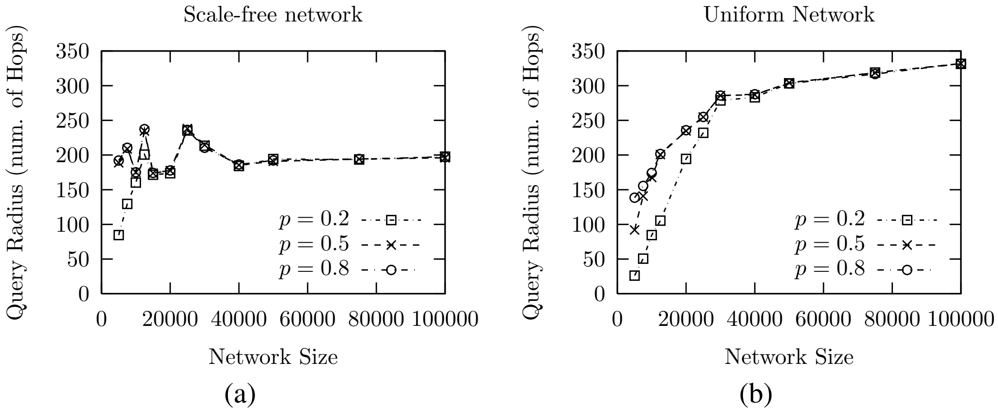
Table 2
Single Group Matching Probability with the given simulation settings

The results regarding the propagation of updates, i.e. update radius and span, are very similar to those of the single tree case. As we already pointed out, the internal structure of each group is a tree, and the updates are propagated within each group according to Algorithm 4, thus obtaining results that are similar to those concerning tree-shaped networks of limited sizes. The difference is that, in the forest of trees setting, updates may need to be propagated to neighboring groups. Once a neighbor is contacted, the update is propagated to its internal nodes, but is not sent to other groups. Thus, the number of nodes affected by an update message depends on the degree of connectivity of groups: if groups have many neighbors, the update will likely be propagated inside these neighbors. However, if groups are made up of many nodes, an update is unlikely to change the bit vectors associated to the group external links, and in this case the update will *not* be propagated outside the group (see Sec. 5.4).



(a) (b)

Fig. 17. Recall for the scale-free(a) and the uniform distributed (b) networks, for $n = 0.2\%$



(a) (b)

Fig. 18. Query radius for the scale-free(a) and the uniform distributed (b) networks, for $n = 0.2\%$

Sensitivity Analysis

In order to support the conclusions derived from the simulation experiments described above, we analyze the contribution of different algorithm parameters to the simulated performance measures. In particular, we perform a so-called 2^F *factorial design* (Jain, 1991), which consist of the following steps:

- Identification of a set of F simulation parameters, called*factors*, whose effect on different measures of performance is to be evaluated. In particular, we consider the maximum number of query hops, precision and recall as the measures of performance of interest.
- Definition, for each parameter, of two different values, a *high* and a *low* value;
- Execution of a total of 2^F simulation experiments, one for each combination of the parameter levels. For each simulation experiment, we record the resulting performance measures, which are called *effects*.

Parameter name	**symbol**	**Levels**	
		-1	1
Total number of nodes in the whole network	N	2000	8000
Probability that a match is found at a node	p	0.04	0.64
Number of attributes for each resource	a	1	4
Number of bits in the RI	k	16	32
Maximum number of nodes in a group	g	10	200
Probability that each node is connected to another node from a different group	l	0.005	0.02

Table 3
Parameters and levels for the regression model

It is the to build a linear regression model in order to compute the effect of each single factor; this is extremely useful in order to identify which factors have the most influence on the observed results.

In our case, we consider the parameters and levels shown in Table 3.

We analyze the linear models describing the relationship between the parameters of Table 3 and the performance metrics Query radius, Precision and Recall. We consider linear models where only the first order contributions are taken into account: that is, we consider linear models of the form:

$$Y = q_0 + q_N x_N + q_g x_g + q_l x_l + q_p x_p + q_k x_k + \textit{residuals}$$

where Y is the performance metric of interest (query radius, precision or recall), x_N, x_g, x_l, x_p and x_k are the levels for factors N, g, l, p and g respectively, and q_0, q_N, q_g, q_l, q_p and q_k are the coefficients which are computed by the model. The term *residuals* contains the cumulative contribution of higher order coefficients, i.e., those describing the contribution of higher order interactions between factors.

Using the R statistical package (Ihaka and Gentleman, 1996) to solve the linear regression model we obtain the results shown in Table. 4.

Query Radius (Hops)			**Recall**			**Precision**		
Param.	**Value**	**Sign.**	**Param.**	**Value**	**Sign.**	**Param.**	**Value**	**Sign.**
q_l	-12.455	no	q_l	0.045101	**yes**	q_l	0.001799	no
q_N	163.230	**yes**	q_N	0.046300	**yes**	q_N	0.006807	no
q_k	-6.215	no	q_k	-0.005999	no	q_k	0.057126	**yes**
q_p	126.313	**yes**	q_p	0.134093	**yes**	q_p	0.095203	**yes**
q_g	-171.852	**yes**	q_g	0.048476	**yes**	q_g	0.012744	no
q_0	226.292	**yes**	q_0	0.844460	**yes**	q_0	0.698430	**yes**

Table 4
Results of the regression model

The *Param.* columns contain the name of the regression coefficient; those labelled *Value* contain the coefficient estimated value. Finally, the columns labelled *Sign.* indicate whether the coefficient is statistically different from zero; if the content of the column is “yes”, then the 95% confidence interval for that parameter value does not contain 0. This means that the performance measure depends on the value of the corresponding factor. If the content of the *Sign.* column is “no”, then the coefficient is *not* statistically different from zero, thus we cannot assume that it influences the result.

From the data shown in Table 4 we find a confirmation of behaviors already observed in the simulation model. We note that the *query radius* (maximum distance in hops from the originating node) of a query message is positively correlated with the number of nodes N in the system, on the probability p that a node matches the query. Both parameters are directly related to the (absolute) number of matches found in the network. More matches implies that a query message has to travel further from the originating point to get most of the possible matches (this will be analyzed further in the following). The query radius is negatively correlated with the group size g, because inter-group links contribute to shorten paths

The recall is positively correlated, among others, with the inter-group link probability l and with the group size g. Again, this is due to the query routing algorithm: larger values of l means that each node is more likely to be connected to other groups so that query messages are more likely to be forwarded to neighbor groups; larger values of g implies that each group contains more nodes. The query propagation algorithm is guaranteed to locate *all* matches within a single group, while it is not guaranteed to locate all matches if they are locate in different groups.

Finally, the precision is correlated with the size of the RI and with the match probability p. It is obvious that larger RI results in better routing accuracy; moreover, larger values of p imply a greater resource density which again results in better accuracy as was already shown in Fig. 4 for the single tree case.

6 Conclusions

In this paper we studied the problem of resource discovery in dynamic Grids by means of P2P techniques. We considered systems where peers hold a set of local resources with associated attributes. Attribute values vary over time, and this makes most of the existing P2P query algorithms not adequate. Users can locate resources by performing range queries over the set of all attributes. We described a routing strategy based on bit-vector RI, which can be used to route queries towards nodes of the system where matches are likely to be found. Moreover, the bit-vector indices can effectively be updated when attribute values change. We showed that the update and query propagation algorithms are efficient since they do not propagate messages over the whole network.

We applied bit-vector indices to two different peer topologies. The first one is a simple topology based on a single tree, where peers are connected via a tree-shaped overlay network. We then proposed a more relaxed network topology based on a forest of trees: in this network we thus have multiple groups of nodes, internally connected as a tree, while the inter-group connections are arbitrary.

One of the main feature of the solutions presented in this article is that they simplify the resolution of multi-attribute, range queries. Such queries are generally difficult to resolve when using DHTs. Many solutions presented in the literature resolve multi-attribute queries by means of a separate DHT for each attribute type. In these cases, the final result is computed by intersecting the lists of partial results obtained by each sub-query, one for each attribute (i.e., lists of matching resources). Moreover, even if the average complexity of locating a single item by using a DHT is usually logarithmic in the size of the network, the complexity may grow to almost linear in the case of queries spanning very large attribute ranges. This is because many DHT nodes, each responsible for a small range of attribute values, must be contacted sequentially.

Unlike the DHT-based networks, our RI approach is based on a natural partitioning/distribution of the global index to the various peers of the network, simply entailed by the resources that have been assigned to each peer node. For example, we can have one or more peer nodes for each Grid VO. Each node can thus compute its local index on the basis of its own resources, and all the attributes associated with them. Finally, every node is capable of locally resolving all the sub-queries on every type of attribute.

This is one of the main results we achieved, by using RI over a tree-shaped overlay network. This network supports frequent updates of the attribute values of resources, without the need to broadcast changes to all the nodes. Note that this is due to the intrinsic properties of both the proposed protocol and the bitwise RIs.

The second solution we proposed, based on a forest of trees, not only preserves these nice features, but also introduces a hierarchical P2P network that is easier to maintain and manage. One problem of this network is that, even if the RI indices are easy to maintain, they may be somewhat imprecise. Therefore, queries could not able to retrieve all the matching resources, thus obtaining a recall that is less that 100%. However, we showed that, if the inter-group connections of this hierarchical network topology are modeled as a scale-free network–which is a very common case in the Web and in P2P networks–and the average group size is large enough, the system can still achieve a good recall, while the network topology ensures a limited radius of query propagation. This is also due to the average shortest paths, that in this type of networks is logarithmic in the number of nodes.

We used simulation results supported by analytical evaluations in order to assess the performance of the query and update routing algorithms for the single tree and forest scenarios. As performance measures we considered the number of hops of messages, precision and recall of queries, and number of nodes receiving a message. Experimental results show that both in the single tree and forest of trees scenario our proposed RI are very effective in limiting the message span and number of hops.

As future work, we are currently extending the proposed algorithms using histogram indices instead of simple bit vectors, following an approach similar to (Petrakis et al., 2004). This allows us to store also informations on the approximate number of matches, which can be very useful for certain applications.

Another open problem which will be investigated is related to limiting the number of links a message is allowed to traverse before being destroyed. In this case users may be unable to get the full list of resources satisfying a query, as potentially useful resources may be beyond the horizon of messages. Clearly, a tradeoff between the value of the time-to-live counter and ability to recall a significant fraction of resources needs to be identified.

References

Andrzejak, A., Xu, Z., 2002. Scalable, efficient range queries for grid information services. In: P2P '02: Proc. of the Second Int. Conf. on Peer-to-Peer Computing. IEEE Computer Society, Washington, DC, USA, p. 33.

Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R., Stoica, I., 2003. Looking up data in p2p systems. Commun. ACM 46 (2), 43–48.

Barabási, A.-L., Reka, A., 1999. Emergence of scaling in random networks. Science 286 (5439).

Bharambe, A. R., Agrawal, M., Seshan, S., 2004. Mercury: Supporting scalable multi-attribute range queries. In: Proc. ACM SIGCOMM 2004 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication. ACM Press, pp. 353–366.

Cai, M., Frank, M., Chen, J., Szekely, P., 2003. Maan: A multi-attribute addressable network for grid information services. In: GRID '03: Proc. of the 4th Int. Workshop on Grid Computing. IEEE Computer Society, Washington, DC, USA, p. 184.

Crainiceanu, A., Linga, P., Gehrke, J., Shanmugasundaram, J., 2004. P-tree: a p2p index for resource discovery applications. In: WWW Alt. '04: Proc. of the 13th Int. World Wide Web conference on Alternate track papers & posters. ACM Press, New York, NY, USA, pp. 390–391.

Crespo, A., Garcia-Molina, H., 2002. Routing indices for peer-to-peer systems. In: ICDCS '02: Proc. of the 22nd Int. Conf. on Distributed Computing Systems (ICDCS'02). IEEE Computer Society, Washington, DC, USA, pp. 23–33.

Foster, I. T., Iamnitchi, A., 2003. On death, taxes, and the convergence of peer-to-peer and grid computing. In: Kaashoek, M. F., Stoica, I. (Eds.), IPTPS. Vol. 2735 of Lecture Notes in Computer Science. Springer, pp. 118–128.

Ganesan, P., Yang, B., Garcia-Molina, H., 2004. One torus to rule them all: multidimensional queries in p2p systems. In: WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases. ACM Press, New York, NY, USA, pp. 19–24.

Gnutella, 2006. Gnutella protocol development. http://rfc-gnutella.sourceforge.net/.

Ihaka, R., Gentleman, R., 1996. R: A language for data analysis and graphics. Journal of Computational and Graphical Statistics 5 (3), 299–314.
URL `http://www.amstat.org/publications/jcgs/`

Jain, R., 1991. The Art of Computer System Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley.

Marzolla, M., Mordacchini, M., Orlando, S., 2006a. A P2P resource discovery system based on a forest of trees. In: In Proc. GLOBE'06 - 2nd Int. Workshop on Grid and Peer-to-Peer Computing. To appear.

Marzolla, M., Mordacchini, M., Orlando, S., Feb. 15–17 2006b. Tree vector indexes: Efficient range queries for dynamic content on peer-to-peer networks. In: Proc. of the 14th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP 2006). IEEE Computer Society, pp. 457–464.

Pacini, F., 3 Feb. 2005. JDL attributes specification. EGEE Document EGEE-JRA1-TEX-555796-JDL-Attributes-v0-1.

Pandurangan, G., Raghavan, P., Upfal, E., Aug. 2003. Building low-diameter peer-to-peer networks. IEEE J. on Selected Areas of Communications 21 (6), 995–1002.

Petrakis, Y., Koloniari, G., Pitoura, E., Aug. 29–30 2004. On using histograms as routing indexes in peer-to-peer systems. In: Ng, W. S., Ooi, B. C., Ouksel, A. M., Sartori, C. (Eds.), DBISP2P. Vol. 3367 of LNCS. Springer, Toronto, Canada, pp. 16–30.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S., 2001. A scalable content-addressable network. In: Proc. SIGCOMM '01. ACM Press, New York, NY, USA, pp. 161–172.

Rowstron, A. I. T., Druschel, P., 2001. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware 2001: Proc. of the IFIP/ACM Int. Conf. on Distributed Systems Platforms Heidelberg. Springer-Verlag, London, UK, pp. 329–350.

Spence, D., Harris, T., 2003. Xenosearch: Distributed resource discovery in the xenoserver open platform. In: HPDC-12: Proc. Twelfth IEEE Int. Symposium on High Performance Distributed Computing. IEEE Computer Society, pp. 216–225.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., Balakrishnan, H., 2003. Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. 11 (1), 17–32.

Talia, D., Trunfio, P., 2003. Toward a synergy between p2p and grids. IEEE Internet Computing 7 (4), 94–96.

Zhao, B., Kubiatowicz, J., Joseph, A. D., April 2001. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB Technical Report UCB/CSD-01-1141, Univ. of California Berkeley, Electrical Engineering and Computer Science Department.