*Norwegian University of Science and Technology*
*Technical Report IDI-TR-08/2007*

# Schema-Assisted XML Querying in Unstructured P2P Systems

Christos Doulkeridis[1], Kjetil Nørvåg[2], and Michalis Vazirgiannis[1]
[1]Dept. of Informatics, AUEB, Athens, Greece
{cdoulk,mvazirg}@aueb.gr
[2]Dept. of Computer Science, NTNU, Trondheim, Norway
Kjetil.Norvag@idi.ntnu.no

ISSN: 1503-416X

**Abstract**

While XML emerges as the de-facto standard for contents available on the Web, centralized approaches for searching the available information is deemed inappropriate at Web scale. Unstructured peer-to-peer (P2P) systems appear as a promising alternative to delve with scalability limitations, hence efficient querying of data stored in XML databases that reside on peers becomes an important challenge. In this paper, we propose a new summary caching method for improving the efficiency of P2P XML query processing. Our approach is based on caching parts of XML schemas along the query path, to enable subsequent *jumps* to remote peers storing content relevant to the query. We evaluate the performance improvements of our search strategy, and its variants, in terms of completeness of the search and reduced latency. The results show that our approach can significantly enhance a naive query mechanism, such as flooding, and improve a baseline summary caching technique that does not exploit the hierarchical structure of the schema.

**Keywords:** Peer-to-peer, XML schema, XML query processing, summary caching, query routing

# 1 Introduction

With the advent of the WWW and the extensive usage of information systems in almost every industrial endeavor there is an overwhelmingly increasing number of information/content providers. The main challenge for these providers is to enable users (both humans as well as computer programs, such as crawlers or agents) to find and retrieve relevant information efficiently. Previously, this requirement was met by organizing the information in large centralized databases.

Recently the vast majority of this information is (or can be made) available on the Web and one can rely on search engines to crawl and index the data. However, this approach is not truly scalable, if the high rate of new content creation is considered. Moreover this is not an appropriate solution for the majority of data, as in the case of data stored in what is called the *hidden* or *deep Web*. In order to tackle these issues, peer-to-peer (P2P) systems emerge as an interesting paradigm, particularly suitable for managing and accessing the vast amounts of data published by a large number of providers. P2P systems constitute an attractive solution as they offer desirable features, such as high degree of autonomy, low administration cost and intrinsic failure tolerance.

In the recent years XML has emerged as the de-facto standard format for information exchange on the Web. A variety of technologies, from Web services, to e-commerce and e-business applications, and further to protocols like WSDL and SOAP, or even rich metadata formats (MPEG*) for multimedia content, support this observation. This demonstates that semi-structured data in a P2P network will probably be represented in XML [16]. Another useful application of our research is federated digital libraries.

Thus, reasonably assuming that XML data is available on peers, the challenge is to enable facilities for searching and querying XML data in a P2P context. Two main strategies are feasible then: 1) using a structured P2P system indexing the data in a distributed hash table [2, 9, 24, 27], or 2) using unstructured P2P networks with some efficient routing strategy [7, 10, 15, 17, 22]. In this paper we focus on unstructured P2P systems, and present a novel *summary caching* approach of XML schemas for efficient search and querying of XML data.

A basic query routing mechanism in unstructured P2P networks is *flooding*. However, this technique is not scalable, and in practice, given a specified search cost in terms of number of messages, it limits the search to only a neighborhood of the querying peer. Several techniques have been proposed for reducing the search cost, and one particular class of techniques is those based on summary caching. In this paper, we present a new summary caching approach that is particularly useful in the context of XML data. Our approach is based on *caching (parts of) schemas of XML documents stored at remote nodes*. Schema information is cached at all peers on the query return path. Assuming a query based on path expression (which is the case for both XPath and XQuery), the schemas can be used for determining peers with high probability of matching the query. Thus the query is more directed than flooding, as it can be forwarded to peers that are more likely to contain relevant data. The result is that given a certain search cost, the recall[1] improves significantly in comparison to a basic approach, like flooding. Compared to previous work, our innovation lies in exploiting hierarchical schema information to cache representative summaries of peers' contents based on the query workload. This is an important difference compared to initial work on web caching [8] and recent work on P2P keyword-based caching, since we devise caching techniques that inherently exploit the hierarchical structure of data. Then query routing efficiency increases, by directing the query to peers with relevant schemas.

The main contributions of our work are:

[1] Recall is the fraction of all relevant material that is returned by a search.

- A novel summary caching technique that uses (parts of) cached XML schemas to improve query routing in unstructured P2P networks.
- An algorithm that encompasses different routing variants, enabling *jumps* to topologically remote peers, based on the local cache contents.
- We perform an extensive evaluation of the approach, through large-scale simulations. The results show that our approach: 1) reduces query processing cost and 2) increases the probability of finding the relevant XML data, in a P2P network where we can not afford to search all peers.

The organization of the rest of this paper is as follows. In Section 2, we overview related work. In Section 3, we define the system architecture, the data and query model. In Section 4, we describe schema-assisted P2P querying of XML data. In Section 5, we provide an evaluation of our approach based on a simulator prototype of a P2P system. Finally, in Section 6, we conclude and outline issues for further work.

## 2 Related work

An extensive review on P2P management of XML data can be found in [16]. One of the research challenges identified in this context is indexing XML data and subsequent efficient query routing. This is the issue addressed by our work. While several approaches in P2P XML data management assume schema-less XML documents [15], this is expected to change. So our approach capitalizes on the use of schemas. The lack of schema might have been good enough for previous document-centric XML documents, but will probably not be for data-centric documents. A schema is a very valuable resource in query processing and should always be used when available. Further, the existence of automatic schema generation techniques allows management of schema-less XML data by our approach too, without requirements of explicit schema definition by a peer.

It should be noted that result caching is not considered in our work as an appropriate alternative solution, mainly due to the high associated storage and maintenance cost. In particular, updates of actual results can lead to excessive bandwidth consumption, depending on the size of the cached objects. Therefore we mainly consider index caching mechanisms, an approach similar to the one adopted in [26]. This protocol (DiCAS) organizes peers into disjoint groups and selectively caches index data on the query return path, to improve some limitations of uniform index caching employed by Gnutella. Compared to DiCAS, our work exploits the hierarchical structure of XML data, more precisely its schema, to cache summary information along the query return path and to subsequently support *jumps* to remote peers holding data relevant to the query. Our approach can be enhanced by this protocol, an issue that we will investigate in future work, to harness the merits of both approaches. Another approach that employs query-driven caching, in which similarly to our work the information cached is generated by the query load, is presented in [23].

Furthermore, in most P2P systems queries are keyword-based, thus supporting only exact matching, that usually does not accommodate the user's needs or requirements. It is therefore important to use more expressive and powerful languages (like XML) that go beyond keyword matching, exploiting the (hierarchical) structure of the data [16].

In the following, we first review related work in XML query processing in unstructured P2P systems, which is more tightly connected to our approach, we then proceed to briefly overview existing related research in structured P2P systems, and finally we overview improvements to basic search in unstructured P2P networks.

## 2.1 XML query processing in unstructured P2P systems

Among the most closely related research papers to our work are [3, 10, 14]. In [10], it is assumed that the XML tags provided by the participating nodes have-system wide defined semantics. This is practically a text indexing approach, which does not exploit the hierarchical information. The system does not try to compensate for semantic heterogeneity, which is considered an orthogonal problem. Whenever a new node joins the system it has to propagate information to every node in the network, which clearly is not scalable. Incremental updates are used to send only changes to data and in the approach mainly stable peers are assumed.

A different family of approaches is based on the concept of routing indices [6] and their variants. In [14], the authors propose strategies for routing and query processing in unstructured P2P systems. The basic mechanism adopted is a variant of the compound routing index, following the query shipping approach, i.e., no caching is assumed. In the same context, in [3], schema-aware routing indices are used to guide queries following a super-peer based approach. Routing of XML queries in P2P databases is researched in [17]. The authors try to propose a scalable solution with respect to both query processing and data updates. They also discuss the infeasibility of flooding or global index maintenance in a dynamic P2P environment, which motivates our approach.

Koloniari and Pitoura [15] present an approach for summary indexing using multi-level Bloom filters. In their work, they consider hierarchical content-based organization of peers, to subsequently route queries efficiently based on the summaries. In [22], a super-peer approach towards a P2P XML database system is presented, where peers are organized in a hierarchy, too. Different data schemas are supported and peers are clustered together based on schema-similarity, to reduce the querying cost. Query routing is based on query propagation up to the root of the hierarchy.

In previous work [7, 19], we have studied the issue of summary caching for improved query processing in unstructured P2P systems. In [19], taxonomy caching is employed for data indexed by a taxonomy. In [7], we presented the notion of schema caching for XML data.

## 2.2 XML query processing in structured P2P systems

Another paper closely related to our ideas is [9]. In this approach the authors propose indexing keywords (which might be tag names etc.) of remote documents. Each peer publishes selected keywords when it joins the system to the peers within its horizon, thus it is more like replication than caching.

In [27], the authors focus on how to locate the data relevant to the queries so that only useful data are involved in query evaluation. Their approach follows the multihop routing approach and encodes the hierarchical information of the XML data into the overlay network, so that routing keys can be hierarchical XML path expressions. In [11], an approach for XML data location using multiple hierarchically organized indexes is presented. Queries are mapped to more specific queries to find results to overcome the limitations of exact-matching. XPath query processing following a structured P2P approach is studied in [2]. XML fragments are put into a distributed hash table (DHT), namely Chord, and the system also uses replication for more efficient processing. Skobeltsyn *et al.* [24] follow an approach where they index all possible paths that occur in *any* document, then use these paths as keys for publishing in P-Grid. They use caching of paths to improve performance by ”remembering” peers that previously returned results. Decision on what to cache is based on statistics (estimates) that each peer collects.

## 2.3 Search in unstructured P2P systems

Basic search techniques in unstructured P2P systems include flooding and random walk strategies. For an analysis of these techniques we refer to [12]. We will not focus further on random walks in the rest of this paper, even though our approach can enhance random walks in a straightforward way.

Several papers describe improvements [28] to the basic flooding strategy using *query jumps* (direct contact with remote peers known or believed to contain relevant information) to remote peers [18, 25]. In the approach described in [18] the object location is also stored in the return path and query jumps can be employed. However, only one answer is assumed for a given search key. In our case, we can have many results and provide mechanisms to find the most appropriate query matches. Sripanidkulchai et al. [25] also present an approach where they use *shortcuts* (direct links to remote peers) which can be used for jumps. Shortcuts are made based on successful previous queries and are not associated to the actual queries (i.e., a shortcut with a particular ranking is used for all queries). This is different from our approach, where we relate summary information to actual information contents. In [4], a dynamic network adaptation mechanism is used, which is in a sense similar to our approach. Finally, in [29], an approach that combines a structured and an unstructured overlay is presented for enhancing P2P search.

# 3 Background

In this section we define the system architecture, the data- and query models.

## 3.1 System architecture

We consider a system of peers connected in an unstructured (Gnutella-like) P2P network. A peer $P$ that joins the P2P network first establishes connection to one or more peers, as part of the basic P2P bootstrapping protocol (the actual protocol depends on the variant of unstructured P2P network, possible techniques include use of *known peers* as well as multicasting). Thus initially, peers are only aware of their $N_n$ immediate neighbors, where $N_n$ is determined as a function of the basic P2P network creation and peer-join strategies.

## 3.2 Data model

The peers in the system store data that is searchable by other peers in the network. This data is stored as XML documents that can be stored either as files or in a database, the only requirement is that it should be possible to query these documents as will be described shortly. Although documents stored as files (and possible also XML documents stored in databases) might have a filename, in a system-wide context the file names are not of interest as our approach is data-centric[2]. XML documents may or may not have an associated description using XML Schema or a DTD. Although in the past it was common with only a DTD at most, for new data-centric applications based on XML the use of schemas will be the rule (or can be automatically generated).

In a P2P context all peers are autonomous. This means that there is no global schema or authority that can control or verify schemas that are used. Thus we expect that on each peer a number of schemas are used. In the general case schemas on different peers will not be related, but in many cases peers will use standardized schemas, which is necessary in the case of communication and, e.g.,

[2]Data-centric XML documents are typically documents meant for computer consumption, while document-centric XML documents are typically meant for human consumption, like books, papers, etc.

e-Business. It should be mentioned that although our approach is most useful when XML schemas are used, we expect that a certain fraction of documents will still be created without an associated schema. In order to improve query efficiency also for queries involving these documents, a ”pseudo DTD” can be created that is compliant with the current instance of the document.

### 3.3 Query model

There exists a number of approaches to query XML documents. This includes standardized languages like XPath and XQuery, as well as vendor-dependent languages and SQL-extensions (which is the typical approach in commercial DBMSs). Most of these approaches have in common that they employ XPath (or a subset of XPath) in order to filter out relevant elements from the XML documents before further processing. Our system-wide query model is also based on XPath.

The most important aspect of XPath is the notion of *location paths* (LP). Example of location paths are `/person/address/city` and `//address/city`. It is intersting to note about location paths that a path $P_1$ that is a prefix of a path $P_2$ will match a superset of the elements matched by $P_2$. For instance, the number of elements matched by `/a/b` will be the same or larger than those matched by the more specific path `/a/b/c`.
*Example:* `/person/address` will match all `person` elements containing an address element, while `/person/address/city` will only math those person elements that contain an `address` element containing a `city` element.

A location path can also contain predicates, which can be tests on strings as well as on numerical values.
*Example:* `/person/address[city="Athens"]` will match all persons having an address with a city attribute containing *Athens*.

In general, the full XPath language or XQuery will be used at a *querying peer* $P_Q$. However, given a query issued by $P_Q$, only the location path, possibly including predicates, is actually forwarded to other peers. When a query is issued, the following steps take place:

1. The local query processor extracts the location path *LP* from the query.

2. *LP* will be forwarded to appropriate peers (as will be described in more detail in the following sections).

3. When a *LP* is received at a peer, it will be applied to XML documents having a schema matching the *LP*. The result of applying the *LP* is a set of elements *E*. The elements *E* are returned to $P_Q$.

4. $P_Q$ will receive a set of *E*, one for each peer having matching documents. In the case of XQuery additional manipulation might be performed, in order to generate the final results.

In the description above we have assumed a *data-centric* application context where all matching data of contacted peers will be retrieved and be part of the process of generating the result. However, we note that in *document-centric* application areas, only a subset of the matching documents will be needed and retrieved. Which document to retrieve is for example based on a ranking decision. For these application areas the identifiers and relevant metadata of the documents containing matching elements are returned, instead of the actual elements. In document-centric application areas also the whole documents and not only individual elements might be needed in the final result. Our proposed techniques are equally applicable for both data- and document-centric application areas, but for simplicity of presentation we will assume a data-centric context in the rest of this paper.

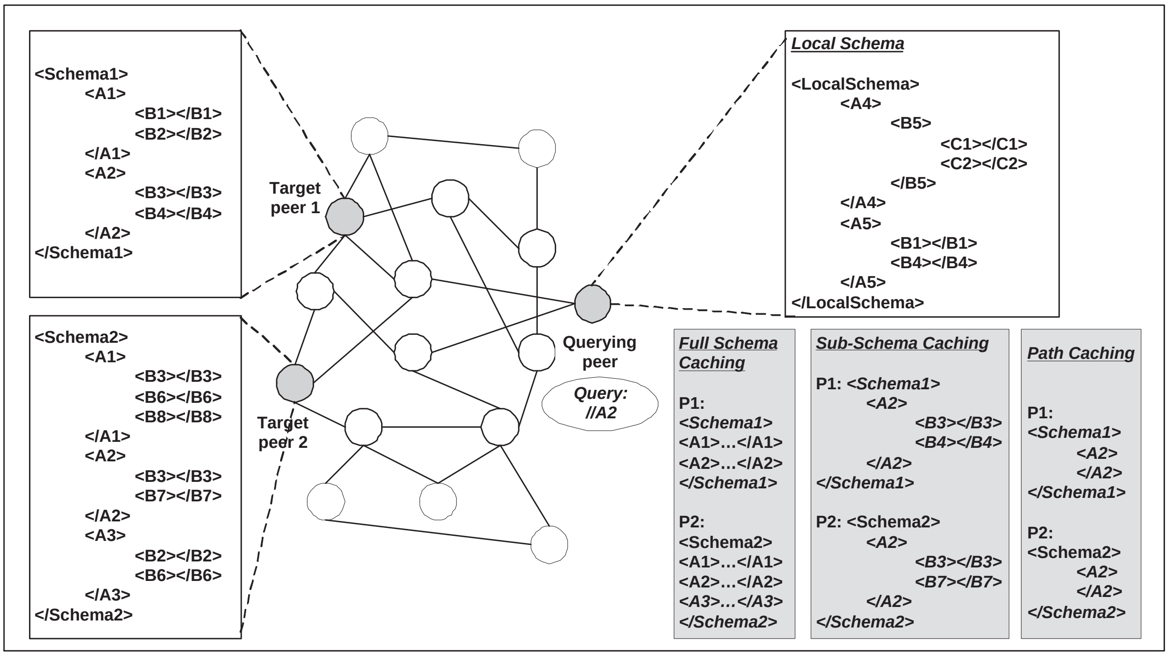



Figure 1: Schema caching variants (*full-schema*, *sub-schema* and *path caching*) as result of a query for `//A2`.

### 3.4 Basic P2P querying

In the basic case, when no performance-improving technique like caching is employed, querying XML documents in an unstructured P2P system is performed as follows: the query $Q$ originating from the querying peer $P_Q$ is forwarded to other peers, denoted *remote peers*. The process of deciding to which peer(s) to forward the query is called *query routing*. Attached to the query is a time-to-live (TTL) value which is initialized by the querying peer. The TTL is decremented each time the query is forwarded, and when it reaches zero the query is not propagated further.

The basic query routing algorithm can be simply flooding or random forwarding, possibly employing more sophisticated techniques like routing indices. The peers that can be reached at a particular time from $P_Q$ constitute the *query horizon* of $P_Q$.

All remote peers receiving the query execute the query locally and forward it to one or more neighbors. Local query execution is performed by applying the received location path expression on all documents marked as shared/searchable. As described above, then either matching elements or document identifiers are returned to $P_Q$.

## 4 Schema-assisted P2P querying of XML data

The problem of using the basic approach for P2P querying as described above is the high cost associated with flooding. The result is that in order to find as many results as possible, a very high number of peers *potentially* having relevant data have to be contacted. In order to have a more targeted query forwarding, we employ information retrieved during past queries to better decide on candidate peers.

One particular kind of information from previous queries is the actual results, i.e., after receiving the *query results* from a number of peers, the querying peer can cache the query results for more efficient processing of future queries as described in, e.g., [1, 21]. An expiration time (presumably with a low value) is attached to the result, to ensure that the results will be discarded after a certain amount of time. In particular for static contents and stable peers, the result caching technique can significantly reduce the search cost. However:

- The results might soon get invalid/stale, and caching the results might incur a considerable storage cost. Thus the results can only be kept for a certain amount of time, limiting the usefulness of the caching.
- The stored results are only useful when the *exact* same query is issued several times.
- Result caching in itself will not improve the probability of finding contents outside the query horizon (note that this can be improved on, by allowing query results to be used also for queries from remote nodes, i.e., not only for local query processing).
- The associated storage and maintenance cost of result caching is high. Even though secondary storage media is getting cheaper, result caching in a dynamic P2P system is not a wise design decision, since the system becomes vulnerable in terms of bandwidth consumption, depending on the rate of updates and the size of updated objects.

An alternative to result caching is to instead cache summary information of contents located at remote peers. This summary information can be returned together with the query results. Ideally, such summary information should be compact and robust to changes of the actual contents.

Our approach is to cache (parts of) the *schema* of remote data (instead of, for example, caching summary based on actual contents), as illustrated in Fig. 1. In the case of XML, a query will involve a path as described in the previous section, and information about possible paths can be found in the schema. Thus having the schema cached locally can be used to know which peers most likely have data that matches the query. In the most extreme case (although not likely in practice), when all schemas from all peers would be available, the exact set of peers matching a query can be found before query forwarding. Notice that DTDs can be employed in cases where they are available, instead of schema.

The advantage of our approach is that even if contents of an XML document change, the schemas in general do not change. It is also the case that usually when a new document is created, it will belong to an existing schema. It should also be noted that our approach can be employed in combination with result caching. This is in particular useful for very frequent queries, which then only have to be reissued at regular times. When our approach is used for query routing, the query horizon is also extended in a more robust way than when using result caching. It will also gradually be extended with time as schema information is further distributed. Essentially both schemas and DTDs can be viewed as a kind of high-quality summaries.

We will now describe in more detail our new approach for query routing. The main contributor to increased query routing efficiency is the XML schema cache (XSCache), a novel structure for efficient maintenance of remote schema information.

## 4.1 Schema-assisted query routing

Query routing is the process of deciding where to forward a query, performed by all peers that are involved in query routing, i.e., both the querying peer as well as intermediate peers. In both cases,

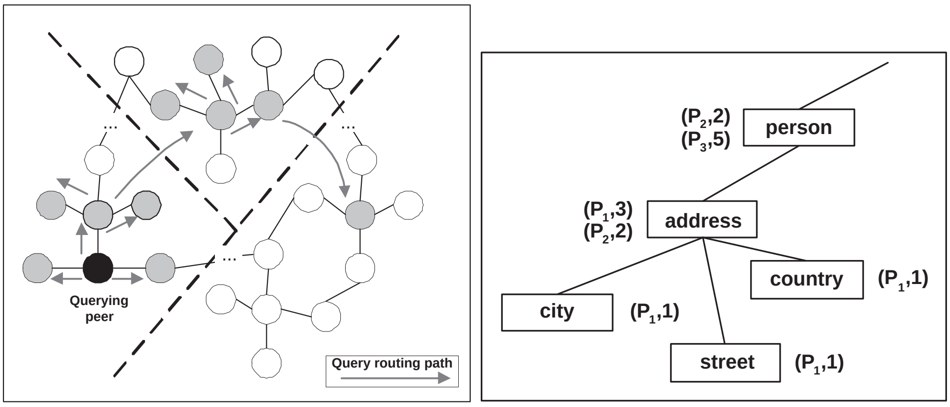



Figure 2: Left: Query routing employing jumps (dashed lines show remote parts of the P2P network, while grey-colored peers are the ones contacted by the query). Right: Tree representation of (parts of) schemas cached from different peers.

when a lookup in the XSCache for schemas matching the XPath expression gives as result a set of matched peers, the query can be forwarded directly to one or more of these peers. This forwarding is called a *jump*. Note that unlike routing indices that only maintain information of the neighborhood and are used to choose appropriate neighbor peers for query forwarding, information about contents at very remote peers (also beyond the query horizon) can be contained in the XSCache. In the case when a peer does not find a match in the XSCache, the query is forwarded using the basic query routing algorithm. Note that even if a query match is found at a peer and results returned to the querying peer, the query is further forwarded until the TTL reaches zero. This is because ideally in a data-centric approach, *all* relevant results to a query need to be retrieved. For an illustrative example of query routing assisted by jumps see Fig. 2 (left), where grey-colored peers are the ones contacted by the query. The figure explains how the query horizon is extended; because of the jumps and the continuation of routing at remote locations, contents of remote XSCaches can be queried and potentially result in new jumps.

In general, a lookup in the XSCache returns $c$ peers having a schema matching the query, and the query is forwarded to $k$ of these peers and in addition to a subset of the neighbors. When $k < c$ a decision has to be made to which of the peers to forward the query. Our approach is to rank the candidate peers based on the number of XML elements they contain that match the cached schema. Those that have the highest number are considered most useful for the query and are more appropriate to answer the query. The value of $k$ is an important parameter, and different strategies for determining $k$ results into a number of *XSCache routing variants*. Assuming $N_n$ to be the average connectivity degree (and let $n = N_n - 1$) for the P2P network then the query is routed to the $k$ highest ranked peers in the cache and to $n - k$ randomly selected neighbors (presented formally in Algorithm 1):

1. *XSCacheB:* $k$=1, select the highest ranked peer from the cache
2. *XSCacheNB*: $k$=$n$, select the $n$ highest ranked peers from the cache
3. *XSCacheN2B*:$k$=$n/2$, select the $n/2$ highest ranked peers from the cache
4. *XSCacheAll*: $k$=$c$, select all peers from the cache

When forwarding the query there is also the issue on how much the TTL should be decremented. We reduce TTL by 1 during jumps, so as to enable the continuation of flooding at remote locations. This means that given a particular TTL the total flooding cost is in the same order whether jumps are performed or not. Nevertheless we emphasize that because of the jumps, more peers will be contacted, since less messages are wasted by reaching already contacted peers. We study the comparative performance of the best XSCache variants experimentally in Section 5.

### 4.2 Distributing schema information

The basic mechanism for distributing schema information is by piggybacking the appropriate schema(s) with the result of the query. The query results is routed to the querying peer through the return path, possibly involving jumps. The reasons for this are: 1) to enable caching the schema(s) at peers on the return path, and 2) to make it possible for these intermediate nodes to validate or invalidate their routing tables. In particular jumps are performed because the peer from which the jump is initiated has information about the destination peer containing data related to the query. This information is kept in the XSCache and has a validity time. By routing the information back, the peer can reinitialize the time counter (because it knows the entry is valid). It is also possible to return an invalidation message indicating that the destination peer does not have any data matching the query anymore, if that is the case.

### 4.3 XML schema caching

The XSCache is a fairly traditional cache using one of the traditional cache replacement algorithms. In addition, each item has an associated expiration time, so that if the validity of the schema with respect to the associated peer has not been confirmed within a certain time, the schema will be removed from the cache.

When caching a schema based on query results, we have the option of caching the whole schema or just the part of the schema relevant for the current query. The first approach is called *full-schema caching* and it is considered an aggressive caching technique, while the second is called *sub-schema caching*. For completeness we also mention that an even coarser caching can be performed: *path caching*. In a path caching approach only the actual path in the query is cached, i.e., not the subpaths being part of the schema.

The three alternatives are illustrated in Fig. 1. The figure shows the contents of the XSCache after a query for `//A2`. The grey-shadowed areas of the figure show the contents of XSCache in case of: 1) full-schema caching (the whole schema is cached), 2) sub-schema caching (only the part of the schema that contains `//A2` is cached), or 3) path caching (only the path ending with `//A2` is cached).

Which of the three approaches (path caching, full-schema caching, and sub-schema caching) to use depends on applications and query patterns. As is quite obvious and will also be evident from the experimental results presented in Section 5, path caching will perform worse and the savings in communication and memory is only marginal. Full-schema caching has the advantage of supporting a larger number of different future queries. However, the added communication cost and memory usage can be significant, so in the general case when no assumptions can be made regarding schema sizes and access pattern, sub-schema caching is a safer approach, thus it will be the context of the following description.

The items in the cache, the (sub-)schemas, are represented as a tree. In order to ensure efficient access and use of memory, schemas from different peers are merged when possible. This is illustrated in Fig. 2 (right). Each node in the tree is annotated with a set of tuples *(P, N)*, where $P$ is a peer iden-

| Symbol | Description | Range of values | Default value |
|---|---|---|---|
| Topology | Topology type (con.degree) | GT-ITM(7),GRID(8) | GRID(8) |
| $N_P$ | Number of peers | 2000...8000 | 2000 |
| $N_S$ | Number of schemas | 100...2000 | 500 |
| $N_D$ | Number of data items | 100000...400000 | 50$\times N_P$ |
| $N_Q$ | Number of queries | - | 100 |
| $q_p$ | Fraction of querying peers | 0.2...0.5 | 0.2 |
| $a$ | Query distribution parameter | 0.6...2 | 1.4 |
| $C_S$ | Cache size | - | 50 |

Table 1: Symbols, descriptions and default values used in the experiments.

tifier and $N$ is the number of leaf elements matching the path from the root to this node in the schema tree. The value of $N$ is used in the ranking of peers as described above. The part of the tree illustrated in Fig. 2 can be the result of a number of queries, including `/person`, `/person/address`, and `//address`. However, a query `/person/address/street` would not generate the tree depicted in Fig. 2. The annotated *(P, N)* tuples are based on information provided together with the schemas from the remote peers.

One important aspect to note about the XSCache is that the values resulting from a query, for example the text string “Athens” in the city element returned from peers, is not cached in the XSCache. The reason is that caching all values would result in high storage cost. Also, caching the values on all peers along the return path (see above) will in general not be beneficial. However, in applications areas where caching the values *is* beneficial, result caching can be applied in addition to schema caching, most typically on the querying peer only.

Representing the schemas as a tree is efficient for queries on the form `/X/Y..`. However, in order to efficiently support queries containing *descendants-of* (i.e., “//”), an XML-index suitable for main-memory processing could be used.

### 4.4 Schema mediation

Because all peers are free to use different schemas even for the same kind of data, schema mediation will be an important issue in such a system. This issue is orthogonal to the topic of this paper, but we should mention that resolving this issue can be performed partly locally by also using techniques to match schemas that do not require perfect match on the path expression. One such technique is mapping and views as described by Cluet et al [5]. Another approach is to use ontologies to map element names.

## 5 Experimental study

In this section, we first present the experimental setup and we then proceed to describe and discuss the results of our experiments. An unstructured P2P network is considered with peers that store XML data that conform to a variety/number of schemas. Some overlap among the schemas that belong to different peers is assumed. A number of peers act as querying peers issuing randomly generated queries to the network to find relevant XML data. Several routing strategies are tested to study their comparative performance.

## 5.1 Experimental setup

We evaluate the performance of the proposed approach through simulations, in order to test its scalability and feasibility. We use a simulator, created by our group and written in Java, for studying the performance of different routing strategies in unstructured P2P networks. Due to reasons mentioned in section 4.3 we will in the experiments use sub-schema caching and path caching, not full schema caching.

Two different topology types are used: 1) a random graph topology created with the GT-ITM topology generator[3], and 2) a grid-like topology of constant connectivity degree which is similar to GT-ITM except that it is more dense, i.e., each pair of neighboring peers share 3-5 common neighbors. We will in the following denote the two kind of topologies as GT-ITM and GRID. We use different network sizes, up to $N_P$=8000 peers. Unless mentioned explicitly, we use the GRID topology with a connectivity degree of 8. We also tried other connectivity degrees in our experiments (not shown here due to lack of space), leading to similar conclusions.

Peer-residing data is described by XML schemas that may have common parts. Also some peers may share a common schema, to resemble the real world case, for instance peers within the same organization. Unless mentioned explicitly, we assume a total of $N_S = 500$ schemas in the network. The total number of available data values in the network is $N_D = 50 \times N_P$, where $N_P$ denotes the number of peers. The allocation of data to peers follows the Pareto principle (also known as the 80-20 rule), i.e., 20% of the peers hold 80% of the total data. We also used uniform allocation to peers. For each data value, a path expression of the peer's local XML schema is used to describe it.

At each simulation experiment a set of $q_p \times N_P$ ($q_p \leq 1$) peers is selected to act as querying peers. Each peer initiates $N_Q$ queries, which are XPath expressions picked from the union of available XML schemas. The queries are randomly generated using the zipfian distribution (with parameter $a$), in order to simulate the users' interests. This is based on previous reports (e.g. see [20]), which state that file popularity on the Web follows a zipfian distribution.

In our simulation study, we assess the performance of different routing strategies using as quality measures:

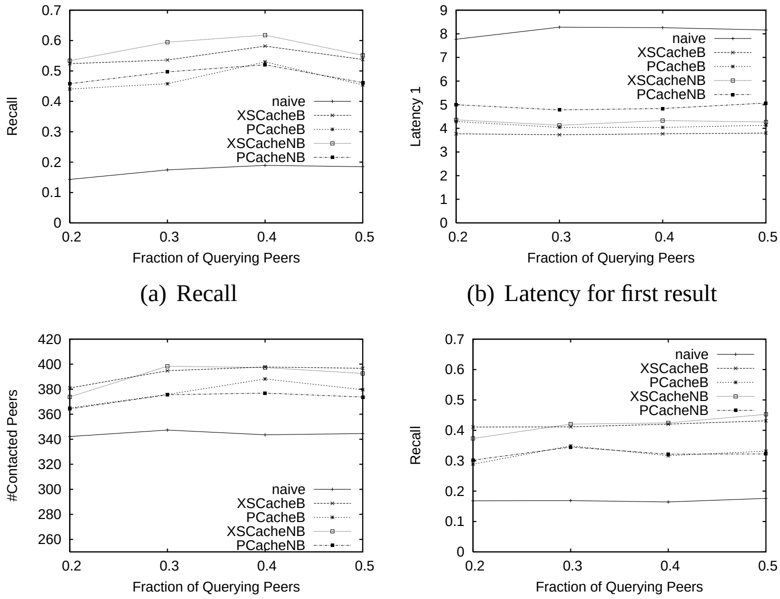
1. Recall, which shows the completeness of the search (in contrast to file-sharing, database applications usually require finding all relevant nodes, and retrieving all answers to a given query [17])
2. Latency for: i) the first result, ii) the first 10 results, and iii) all results, in terms of number of hops required to return the matching XML data to the querying peer
3. Number of contacted peers due to the routing strategy

All measurements are taken after half of the total queries have been issued, in order to eliminate the warm-up effect of the cache. We study the effect of the following parameters on the performance of these strategies: 1) TTL values, 2) network sizes, 3) different topologies, 4) different skew in query distributions, 5) percentage of querying peers, and 6) total number of available schemas in the network.

We use as naive approach two variants of flooding: naive flooding and normalized flooding[4]. In the charts, we present the results of naive flooding only and we refer to this routing strategy as *naive*. We obtained similar comparative results by employing normalized flooding.

[3] Available at: `http://www-static.cc.gatech.edu/projects/gtitm/`

[4] Normalized flooding [13] is a variant of flooding where the query is only forwarded to $d$ neighbors ($d$ is the minimum connectivity degree of any node in the network).




(a) Recall (b) Latency for first result

(c) Number of contacted peers (d) Recall (Uniform allocation)

Figure 3: Measurements from using different number of querying peers, given as the fraction of peers in the network.

In the experiments presented in this section, we present two variants of sub-schema caching (denoted *XSCacheB* and *XSCacheNB*), according to the way the cached schemas are utilized during query routing. We choose to show only two variants, in order to make the charts readable. Moreover, the results of the other XSCache variants do not offer any substantial additional insight. It should be reminded here that with each cached XML schema, also the number of associated data values is kept. Query routing variants are based on how to pick peers to forward the query. A peer is ranked according to the number of data it possesses. The highest ranked peer based on the cached XML schemas is the one that probably keeps the most data values that match the query.

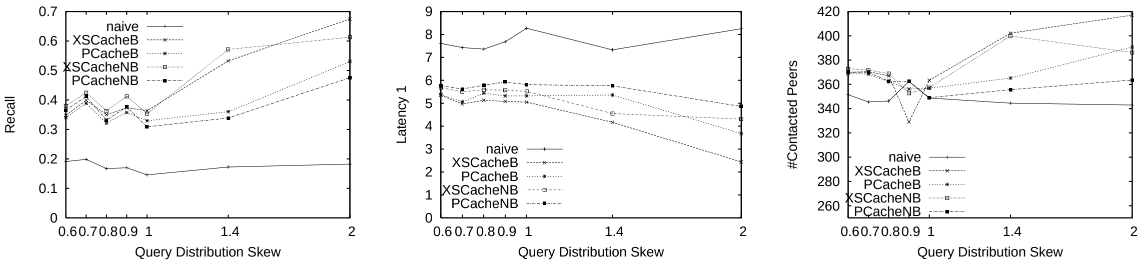
A more sophisticated baseline than *naive* is considered too, where the employed caching mechanism is path caching. We refer to this strategy as *PCache*. In full accordance with the two variants of XSCache, path caching is used in our experiments with two variants: *PCacheB* and *PCacheNB*. The variants of our approach are compared against these two baselines in the experiments.

It should also be stressed that we compare the performance of the aforementioned approaches *using the same number of messages* and the same setup parameters, in order to present directly comparable results. As can be seen, our approach both enhances the naive search strategy and outperforms the baseline search in all experiments.

## 5.2 Experimental results

### 5.2.1 Effect of number of querying peers.

In Fig. 3, we study the effect of increasing number of querying peers in the network. The number of querying peers is increased from 20% to 50% of the total peers. The results show that recall increases with the percentage of querying peers (Fig. 3(a)). In a real P2P system all peers are expected to issue queries for data, so this is a strong argument in favor of the scalability of our approach with the




(a) Recall (b) Latency for first result (c) Number of contacted peers

Figure 4: Measurements from using different skew (represented by increasing values of $a$) in the query distributions.

number of querying peers. Notice also that our approach presents significantly lower latency for the first result compared to the naive approach (Fig. 3(b)). XSCache also outperforms PCache in all other latency-related metrics (first 10 and all results). Another interesting observation comes from Fig. 3(c), namely that XSCache manages to reach more peers than the other approaches, using the same number of messages. In other words, XSCache overcomes some of the problems of a basic routing mechanism related to the structure of the underlying topology, such as the density of the GRID topology.

Finally, we tested uniform allocation of data to peers in Fig. 3(d), which is directly comparable to Fig. 3(a). The comparative results are the same with 80-20 allocation, only the absolute values are lower when the distribution is uniform. We use the 80-20 allocation rule in the rest of the experiments.

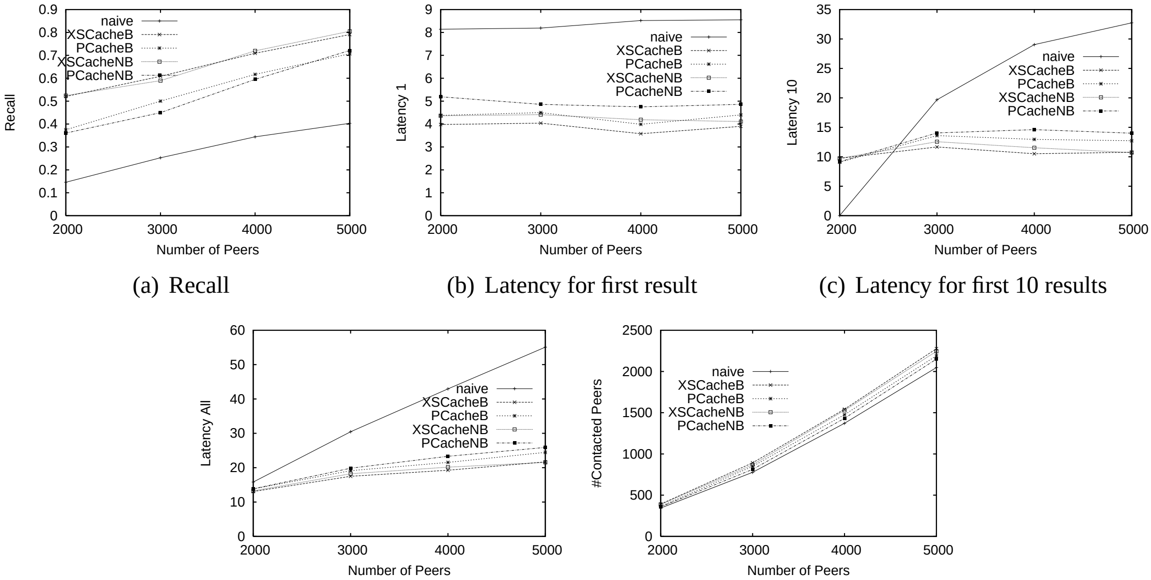
#### 5.2.2 Effect of query distribution skew.

In Fig. 4, we gradually increase the skew in the query distribution, to study the behavior of our approach. As expected, when the query distribution is more skewed, recall increases and latency drops, as a result of many recurring queries. This is due to the fact that the cached XML schemas become useful for more queries. As a consequence, the number of contacted peers increases too. Compared to PCache, XSCache is constantly better in all quality indices.

#### 5.2.3 Effect of network size.

In Fig. 5, the scalability of our approach with respect to the number of peers is illustrated. We stress here that the increasing recall values with network size (Fig. 5(a)) are not expected, yet this is explained due to the fact that we also increase the TTL as we increase the network size. Unfortunately, the TTL does not increase analogously to the number of peers, rather it increases faster, hence recall increases with network size in the chart. The important finding of this experiment is that the higher recall achieved by XSCache relative to PCache (and naive) is sustained as the network size increases.

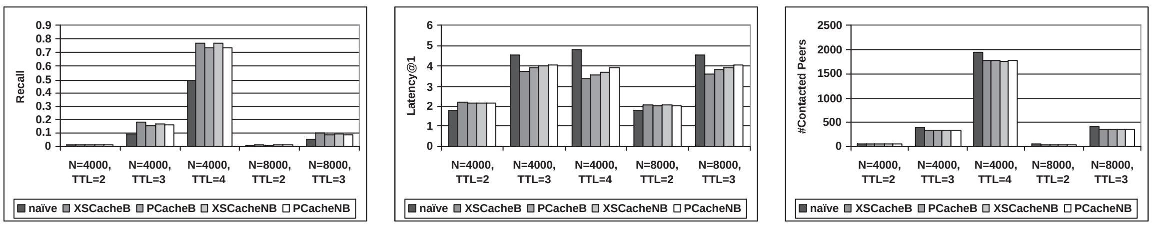
The scalability of the approach is also supported by the latency results, as latency is less sensitive to the increasing TTL values. As can be seen (Fig. 5(b)) the latency for the first result is practically preserved, showing that our approach scales with network size. Further, the latency for the first 10 and for all results increases smoothly with network size. The missing point for naive in Fig. 5(c), and any subsequent latency figures, for 2000 peers is due to the fact that naive does not manage to retrieve 10 results in this setup. In Fig. 5(e), the number of contacted peers is depicted, showing that XSCache manages to contact more peers with the same number of messages.




(a) Recall (b) Latency for first result (c) Latency for first 10 results

(d) Latency for all results (e) Number of contacted peers

Figure 5: Measurements for different network sizes.



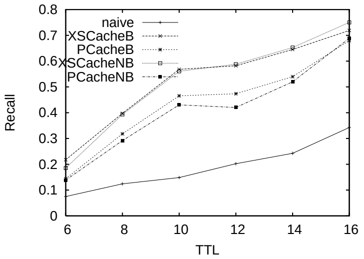

(a) Recall (b) Latency for first result (c) Number of contacted peers

Figure 6: Measurements from using different TTL values in the search, employing GT-ITM topologies of 4000 and 8000 peers and average connectivity degree equal to 7.

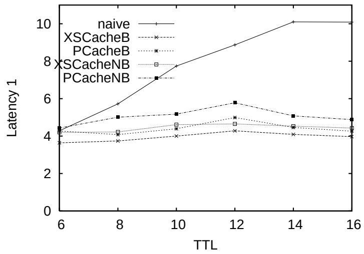
### 5.2.4 Effect of time-to-live.

In Fig. 6, we use the GT-ITM topology and study the effect of increasing TTL values for network sizes of $N_P$=4000 and $N_P$=8000 peers with average connectivity degree 7. We also tested other degrees acquiring similar results. Our approach performs better than the baseline in all cases, showing that it is independent of the underlying network topology. The naive strategy works rather acceptably for GT-ITM, because this topology is relatively sparse, without many common neighbors between peers, thus flooding with sufficiently high TTL values performs relatively well.

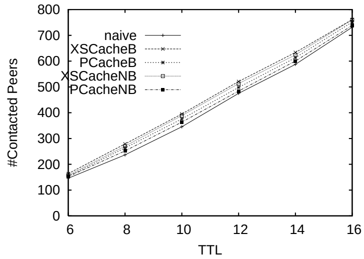
Finally, in Fig. 7, we study the effect of increasing TTL values on the performance of the search strategies using the GRID topology. Obviously recall increases with larger time-to-live values and the same holds for latency for all results. The latency for the first result is stable in our approach, showing that our approach works well also for smaller TTL values, hence smaller network costs.




(a) Recall

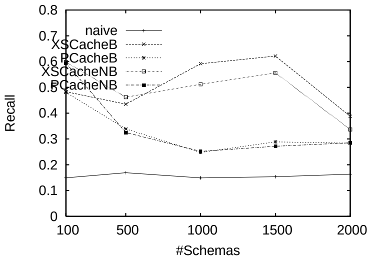



(b) Latency for first result

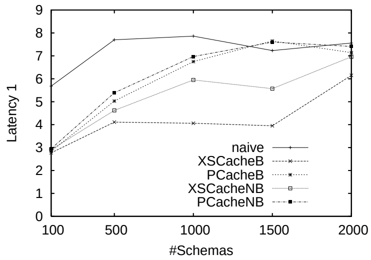



(c) Number of contacted peers

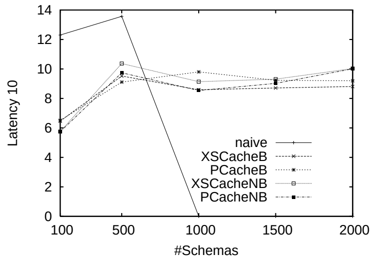
Figure 7: Measurements from using different TTL values in the search, employing the GRID topology.



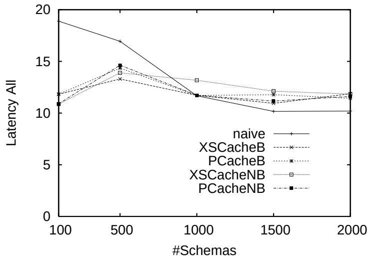

(a) Recall



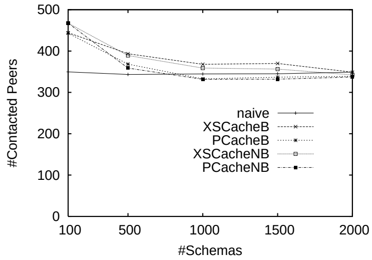

(b) Latency for first result




(c) Latency for first 10 results




(d) Latency for all results




(e) Number of contacted peers

Figure 8: Measurements from using different number of available schemas in the network, employing the GRID topology.

### 5.2.5 Effect of number of schemas.

The effect of varying the number of available schemas in a P2P network of 2000 peers is depicted in Fig. 8. We use up to 2000 different schemas in this experiment, which is a reasonable value since even if data in a P2P system is some orders of magnitude larger than the number of peers, the number of schemas employed is definitely comparable to the number of peers.

Studying recall (Fig. 8(a)), we see that for small number of schemas (100) our approach performs similar to the baseline, but far better than naive. However as the available number of schemas increase, XSCache constantly outperforms the baseline. Notice that this recall does not increase monotonically, as can be seen for 2000 schemas, where the achieved recall admittedly drops, however it still surpasses the baseline. The reduced recall as the number of schemas increase is expected, as in the extreme case that each peer has a completely different schema from other peers, any schema caching mechanism will only benefit queries for this peer only. Moreover, when the number of available schemas decreases, as would be the case when schema mediation is employed, the efficiency of XSCache is not reduced significantly. All in all, with increasing number of schemas, our approach achieves 2-4x times the recall of naive, and 1.5-2.5x times the recall of the baseline.

Regarding latency for the first result, XSCache is constantly better than the baseline, while for the other latency values (latency for 10 results and for all results) XSCache is at least as good as the baseline. The number of contacted peers by our approach is either equal to or slightly higher than the number of peers contacted by the baseline approach. Summarizing, XSCache outperforms its competitors, demonstrating the appropriateness of our approach with increasing the heterogeneity (i.e. number) of available schemas in the system.

### 5.3 Discussion

Concluding, our approach significantly outperforms the baseline search strategies in terms of recall and associated latency (for the first, the first 10 and for all results), while utilizing the same number of messages. In other words, given a certain recall, XSCache needs a lower number of messages to achieve it. Furthermore, our approach increases the number of peers that need to be contacted, with the same search cost.

Comparing the individual variants of XSCache (XSCacheB and XSCacheNB), we conclude that using the highest ranked entry in the cache performs similarly to the approach that uses $n$ entries. As a rule of thumb, we state that even one cached entry (the highest ranked) is enough to achieve high recall and low latency. This conveys that XSCacheB is a suitable/appropriate approach when designing a search protocol for P2P networks similar to the ones tested in our simulations.

## 6 Conclusions and future work

In this paper, we presented a new approach aiming at reducing XML query processing costs in unstructured P2P systems. We introduced the usage of XML schema caching for query routing as well as the underlying mechanisms. We also performed an extensive analysis of the approach through large-scale simulations. The results showed that our approach: 1) reduces query processing cost and 2) increases the probability of finding the relevant XML data, in a P2P network where we can not afford to search all peers.

Plans for future work include integrating actual XML processors in an implementation to develop a P2P XML search engine. We also plan to study in more detail how schema mediation can be optimally implemented in such a system.

## References


[1] B. Bhattacharjee, S. Chawathe, V. Gopalakrishnan, P. Keleher, and B. Silaghi. Efficient peer-to-peer searches using result-caching. In *Proceedings of the 2nd IPTPS*, 2003.

[2] A. Bonifati, U. Matrangolo, A. Cuzzocrea, and M. Jain. XPath lookup queries in P2P networks. In *Proceedings of WIDM'04*, 2004.

[3] I. Brunkhorst, H. Dhraief, A. Kemper, W. Nejdl, and C. Wiesner. Distributed queries and query optimization in schema-based P2P-systems. In *Proceedings of DBISP2P'03*, 2003.

[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P systems scalable. In *Proceedings of SIGCOMM'03*, 2003.

[5] S. Cluet, P. Veltri, and D. Vodislav. Views in a large scale XML repository. In *Proceedings of VLDB'2001*, 2001.

[6] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of ICDCS'02*, page 23, 2002.

[7] C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis. Schema caching for improved XML query processing in P2P systems. In *Proceedings of IEEE P2P'06*, 2006.

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

[9] L. Galanis, Y. Wang, S. Jeffery, and D. DeWitt. Locating data sources in large distributed systems. In *Proceedings of VLDB'03*, 2003.

[10] L. Galanis, Y. Wang, S. Jeffery, and D. DeWitt. Processing queries in a large peer-to-peer system. In *Proceedings of CAISE'03*, 2003.

[11] L. Garcés-Erice, P. A. Felber, E. W. Biersack, G. Urvoy-Keller, and K. W. Ross. Data indexing in peer-to-peer DHT networks. In *Proceedings of ICDCS'04*, pages 200–208, 2004.

[12] C. Gkantsidis, M. Mihail, and A. Saberi. Random walks in peer-to-peer networks. In *Proceedings of INFOCOM'04*, 2004.

[13] C. Gkantsidis, M. Mihail, and A. Saberi. Hybrid search schemes for unstructured peer-to-peer networks. In *Proceedings of INFOCOM'05*, 2005.

[14] M. Karnstedt, K. Hose, and K.-U. Sattler. Query routing and processing in schema-based P2P systems. In *Proceedings of DEXA workshops 2004*, 2004.

[15] G. Koloniari and E. Pitoura. Content based routing of path queries in peer-to-peer systems. In *Proceedings of EDBT'04*, 2004.

[16] G. Koloniari and E. Pitoura. Peer-to-peer management of XML data: Issues and research challenges. *SIGMOD Rec.*, 34(2):6–17, 2005.

[17] N. Koudas, M. Rabinovich, D. Srivastava, and T. Yu. Routing XML queries. In *Proceedings of ICDE'04*, page 844, 2004.

[18] D. A. Menascé and L. Kanchanapalli. Probabilistic scalable P2P resource location services. *SIGMETRICS Performance Evaluation Review*, 30(2):48–58, 2002.

[19] K. Nørvåg, C. Doulkeridis, and M. Vazirgiannis. Taxonomy caching: A scalable low-cost mechanism for indexing remote contents in peer-to-peer systems. In *Proceedings of IEEE ICPS'06*, 2006.

[20] V. N. Padmanabhan and L. Qiu. The content and access dynamics of a busy web site: findings and implications. In *Proceedings of SIGCOMM'00*, pages 111–123, 2000.

[21] S. Patro and Y. C. Hu. Transparent query caching in peer-to-peer overlay networks. In *Proceedings of IPDPS'2003*, 2003.

[22] C. Sartiani, P. Manghi, G. Ghelli, and G. Conforti. XPeer: A self-organizing XML P2P database system. In *Proceedings of EDBT Workshops 2004*, 2004.

[23] G. Skobeltsyn and K. Aberer. Distributed Cache Table: Efficient query-driven processing of multi-term queries in P2P networks. In *Proceedings of P2PIR'06*, 2006.

[24] G. Skobeltsyn, M. Hauswirth, and K. Aberer. Efficient processing of XPath queries with structured overlay networks. In *OTM Conferences (2)*, pages 1243–1260, 2005.

[25] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *Proceedings of INFOCOM'03*, 2003.

[26] C. Wang, L. Xiao, Y. Liu, and P. Zheng. DiCAS: An efficient distributed caching mechanism for P2P systems. *IEEE Transactions on Parallel and Distributed Systems*, 17(10):1097–1109, 2006.

[27] Q. Wang and M. Oszu. A data locating mechanism for distributed XML data over P2P networks. Technical report, CS-2004-45, University of Waterloo, School of Computer Science, Waterloo, Canada, 2004.

[28] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings of ICDCS'2002*, 2002.

[29] R. Zhang and Y. Hu. Assisted peer-to-peer search with partial indexing. In *Proceedings of INFOCOM'05*, 2005.

**Algorithm 1** Peer selection during query routing.

**Require:**

$P_N = \{P_{N1}, ..., P_{Nn}\}$ {Neigbors to consider, $n = N_n - 1$}

$XSCRV \in \{XSCacheB, XSCacheNB, XSCacheN2B, XSCacheAll\}$ {XSCache routing variant}

$P_C = \{P_{C1}, ..., P_{Cc}\}$ {Peers of the $c$ matching entries in XSCache, decreasing rank}

**Ensure:**

$P_F$ {Peers to which the query will be forwarded}

$P_F = \emptyset$

**if** ($c < n$) **then**

  $P_R = (n - c)$ randomly selected peers from $P_N$

**end if**

**if** ($XSCRV = XSCacheNB$) **then**

  **if** ($c \geq n$) **then**

    $P_F = \{P_{C1}, ..., P_{Cn}\}$

  **else**

    $P_F = P_C \bigcup P_R$

  **end if**

**else if** ($XSCRV = XSCacheB$) **then**

  $P_F = P_{C1} \bigcup P_N$

**else if** ($XSCRV = XSCacheAll$) **then**

  **if** ($c \geq n$) **then**

    $P_F = P_C$

  **else**

    $P_F = P_C \bigcup P_R$

  **end if**

**else if** ($XSCRV = XSCacheN2B$) **then**

  **if** ($c \geq n$) **then**

    $h = n/2$

    $P_F = \{P_{C1}, ..., P_{Ch}\} \bigcup$ random selection of $h$ other peers in $P_C$

  **else**

    $P_F = P_C \bigcup P_R$

  **end if**

**end if**