Content-Based Overlay Networks of XML Peers Based on Multi-Level Bloom Filters[1]

Georgia Koloniari, Yannis Petrakis, Evaggelia Pitoura

Department of Computer Science
University of Ioannina, Greece
{kgeorgia, pgiannis, pitoura}@cs.uoi.gr

Abstract. Peer-to-peer systems are gaining popularity as a means to effectively share huge, massively distributed data collections. In this paper, we consider XML peers, that is, peers that store XML documents. We show how an extension of traditional Bloom filters, called multi-level Bloom filters, can be used to route path queries in such a system. In addition, we propose building content-based overlay networks by linking together peers with similar content. The similarity of the content (i.e., the local documents) of two peers is defined based on the similarity of their filters. Our experimental results show that overlay networks built based on filter similarity are very effective in retrieving a large number of relevant documents, since peers with similar content tend to be clustered together.

1 Introduction

Peer-to-peer (p2p) computing refers to a new form of distributed computing that involves a large number of autonomous computing nodes (the peers) that cooperate to share resources and services [17]. P2p systems are gaining popularity as a way to effectively share huge, massively distributed data collections.

In this paper, motivated by the fact that XML has evolved as the new standard for data representation and exchange on the Internet, we assume that peers store XML documents: either XML files that they want to share or XML-based descriptions of the resources and services that they offer. We extend search in p2p, by considering path queries that explore the structure of such hierarchical documents.

Bloom filters have been proposed for summarizing documents (e.g., in [3]). Bloom filters are compact data structures that can be used to support membership queries, i.e., whether an element belongs to a set. In [9], we have introduced multi-level Bloom filters that extend traditional simple Bloom filters for answering path queries.

We show how multi-level Bloom filters can be used to route queries in a p2p system. Each peer maintains a summary of its content (i.e., local documents) in the form of a multi-level Bloom filter. It also maintains one merged multi-level Bloom

[1] This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-32645 DBGlobe project.

filter for each of its links summarizing the content of all peers that can be reached through this link. Using such merged filters, each peer decides to direct a query only through links that may lead to peers with documents that match the query. For scalability reasons, we limit the number of peers that are summarized through the concept of horizons. We also propose a heuristic for choosing through which among more than one qualifying link to route the query.

Furthermore, we show how Bloom filters can be used to build content-based overlay networks, that is, to link together peers with similar content. The similarity of the content (i.e., the local documents) of two peers is defined based on the similarity of their filters. This is cost effective, since a filter for a set of documents is much smaller than the documents themselves. Furthermore, the filter comparison operation is more efficient than a direct comparison between sets of documents. Our experimental results show that overlay networks built based on filter similarity are very effective in retrieving a large number of relevant documents, since relevant peers tend to be clustered together.

In summary, this paper makes the following contributions:

- Extends keyword queries in p2p to path queries by proposing multi-level Bloom-based filters,
- Shows how traditional and multi-level Bloom filters can be used to search in a horizon-based distribution and proposes a search heuristic based on the characteristics of Bloom filters,
- Proposes building content-based overlay network, where a peer connects to peers with similar content where similarity is based on the similarity of their filters. Such networks are shown to be very effective in retrieving a large number of relevant documents.

The remainder of this paper is structured as follows. Section 2 describes the system model, our multi-level Bloom filters and filter distribution using horizons. Section 3 introduces the two different overlay network organizations based on proximity and content-similarity criteria and our similarity metric. Section 4 presents our experimental results. In Section 5, we compare our work with related research and we conclude in Section 6 with directions for future work.

2 Multi-Level Bloom Filters as P2P Routers

2.1 System Model

We consider a system of peers where each peer n_i maintains a set of documents D_i (a particular document may be stored in more than one peer). Each peer is logically linked to a relatively small set of other peers called its *neighbors*. Motivated by the fact that XML has evolved as the new standard for data representation and exchange on the Internet, we assume that peers store XML files: XML documents that they want to share or XML-based descriptions of the available local services and resources.

In our data model, an XML document is represented by an unordered-labeled tree, where tree nodes correspond to document elements, while edges represent direct

element-subelement relationships. Although, most p2p systems support only queries for documents that contain one or more *keywords*, we want also to query the structure of documents. Thus, we consider *path queries* that are simple path expressions in an XPath-like query language.

Definition 1 (path query): A simple path expression query of length p has the form "$s_1\ l_1\ s_2\ l_2\ \ldots\ s_p\ l_p$" where each l_i is an element name and each s_i is either / or // denoting respectively parent-child and ancestor-descendant traversal.

A keyword query searching for documents containing keyword k is just the path query //k. For a query q and a document d, we say that q is satisfied by d, or *match(d, q)* is true, if the path expression forming the query exists in the document. Otherwise we have a *miss*.

A given query may be matched by documents at various peers. To locate peers with matching documents, we maintain at each peer specialized data structures called *filters* that summarize large collections of documents reachable from this peer. The aim is to be able to deduce whether there is a matching document along a particular link by just looking at the filter. To this end, each filter *F(D)* that summarizes a set of documents D should support an efficient *filter-match* operation, *filter-match(F(D), q)*, that for each query q, if there is a document $d \in D$ such that *match(d, q)* is true then *filter-match(F(D), q)* is also true. If the filter-match returns false, then we have a *miss* and we can conclude that there is no matching document in D. The reverse does not necessarily hold. That is, *filter-match(F(D), q)* may be true but there may be no document $d \in D$ for which *match(d, q)* is true. This is called a *false positive* and may lead to following paths to peers with no matching documents. However, no matching documents are lost. We are looking for filters for which the probability of false positive is low.

Bloom filters are appropriate as summarizing filters in this context in terms of scalability, extensibility and distribution. However, they do not support path queries. To this end, we have proposed an extension called multi-level Bloom filters [9].

2.2 Multi-Level Bloom Filters

Bloom filters are compact data structures for probabilistic representation of a set that supports membership queries ("Is element a in set A?"). Since their introduction [1], Bloom filters have been used in many contexts including web cache sharing [2], query filtering and routing [3, 4] and free text searching [5].

Consider a set $A = \{a_1, a_2, \ldots, a_n\}$ of n elements. The idea (Figure 1) is to allocate a vector v of m bits, initially all set to 0, and then choose k independent hash functions, $h_1, h_2, \ldots, h_k$, each with range 1 to m. For each element $a \in A$, the bits at positions $h_1(a), h_2(a), \ldots, h_k(a)$ in v are set to 1. A particular bit may be set to 1 many times. Given a query for b, we check the bits at positions $h_1(b), h_2(b), \ldots, h_k(b)$. If any of them is 0, then certainly b is not in the set A. Otherwise, we conjecture that b is in the set although there is a certain probability that we are wrong, i.e., we may have a false positive. This is the payoff for Bloom filters compactness. The parameters k and m should be chosen such that the probability P of a false positive is acceptable. It has been shown [1] that: $P = (1 - e^{-kn/m})^k$.

To support updates of the set *A,* we maintain for each location *l* in the bit array a count *c*(*l*) of the number of times that the bit is set to 1 (the number of elements that hashed to l under any of the hash functions). All counters are initially set to 0. When a key *a* is inserted or deleted, the counters $c(h_1(a))$, $c(h_2(a))$, ..., $c(h_k(a))$ are incremented or decremented accordingly. When a counter changes from 0 to 1, the corresponding bit is turned on. When a counter changes from 1 to 0, the corresponding bit is turned off.

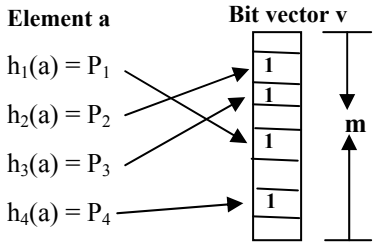

Fig. 1. A Bloom filter with k = 4 hash functions.

For processing a path query using Bloom filters, we check whether each element of the path is matched by the filter. If there is a match for every element, then we conjecture that the path may exist, not taking structure into account.

We consider two ways of extending Bloom filters for hierarchical documents. Let an XML tree *T* with *j* levels, and let the level of the root be level 1. The *Breadth Bloom Filter* (BBF) for an XML tree *T* with *j* levels is a set of *i* +1 Bloom filters $\{BBF_0, BBF_1, BBF_2, \ldots, BBF_i\}$, $i \leq j$. In BBF_0, we insert all elements that appear in any level of the tree. Then, there is one Bloom filter, denoted BBF_i, for the level *i* of the XML tree, in which we insert the elements of all nodes at level *i*. Depth Bloom filters provide an alternative way to summarize XML trees. We use different Bloom filters to hash paths of different lengths. The *Depth Bloom Filter* (DBF) for an XML tree *T* with *j* levels is a set of *i* Bloom filters $\{DBF_0, DBF_1, DBF_2, \ldots, DBF_{i-1}\}$, $i \leq j$. There is one Bloom filter, denoted DBF_i, corresponding to the paths of the tree of length *i*, (i.e., having $i + 1$ nodes), where we insert all paths of length *i*. Note that we insert paths as a whole, we do not hash each element of the path separately; instead, we hash their concatenation.

To implement the filter match for a path query, in the case of BBFs, we first check, whether all elements in the path expression appear in BBF_0. Then, the first element of the path query is checked at BBF_1. If there is a match, the next element is checked at the next level of the filter and the procedure continues until either the whole path is matched or there is a miss. For paths with the ancestor-descendant axis //, the path is split at the //, and the sub-paths are processed at all the appropriate levels. All matches are stored and compared to determine whether there is a match for the whole path.

The procedure that checks whether a DBF matches a path query, first checks whether all elements in the path expression appear in DBF_0. If this is the case, for a query of length *p*, every sub-path of the query from length 2 to *p* is checked at the filter of the corresponding level. If any of the sub-paths does not exist then the algorithm returns a miss. For paths that include the ancestor-descendant axis //, the path is split at the // and the resulting sub-paths are checked. If we have a match for all sub-paths the algorithm succeeds, else we have a miss.

2.3 Filter Distribution Based on Horizons

A query may originate at any peer of the network, whereas documents matching the query may reside in numerous other peers. To direct the query to the appropriate peers, each peer maintains a number of filters. In particular, each peer n_i maintains a *local filter* $F(D_i)$ that summarizes all documents D_i stored locally at n_i and one filter, called *merged filter*, for each of its links. The merged filter for a link e of n_i summarizes the documents that reside at peers reachable from n_i through any path starting from link e. Merged filters are used to direct the query only to peers that have a large probability to contain documents matching the query.

Ideally, the merged filter for each link should summarize the documents of *all* peers reachable through this link. However, this introduces scalability problems. In this case, an update of the content of a peer must be propagated to a huge number of other peers. The same holds for the filters of peers joining or leaving the network. By introducing *horizons*, a peer bounds the number of neighbors whose documents it summarizes. The horizon of a peer n_i includes all peers that can be reached with at most R hops starting from n_i. We call R the *radius* of the horizon.

Definition 2 (Distance): *The distance between two peers n_i and n_j, $d(n_i, n_j)$ is the number of hops on the shortest path from n_i to n_j in the overlay network.*
Definition 3 (Horizon): *A peer n_i has a horizon of R, if it stores summaries for all peers n_j for which the distance $d(n_i, n_j) \leq R$, where R is the radius of the horizon.*

In horizon-based distribution, the merged filter for a link e of n_i summarizes (i.e., merges the local filters) of all peers that are reachable from n_i by a path of length R or smaller starting from e. Figure 2 shows for each link of peer 5, the local filters of which peers are merged at the corresponding merged filter, when $R = 2$. In the case of cycles, the documents of some peers (peer 8 in this example) may be included in more than one merged filter. We describe later, why and how this may be avoided.

In order to calculate the merged Bloom filter of a set of Bloom filters we take the bitwise OR of these Bloom filters. In particular, the merged Bloom filter BF_m of a set $\{BF_1, BF_2, \ldots, BF_n\}$ of Bloom filters is equal to BF_1 *OR* BF_2 *OR* …*OR* BF_n. Similarly, for multi-level Blooms, we take the bitwise OR for each of their levels. Apart from the merged filter, merged counters are also stored, to support updates. Merged counters are produced by adding together the corresponding counters of the set of Bloom filters.

2.4 Join and Update

When a new peer n_k joins the system, it must inform the other peers at distance R about its documents. To this end, n_k sends a message *New($F(D_k)$, Counter)* to all its neighbors, where $F(D_k)$ is its local filter (i.e., the filter summarizing its documents) and *Counter* is set to R. Upon receipt of a *New* message, each peer n_i merges the received $F(D_k)$ filter with the merged filter of the corresponding link. Then, it reduces *Counter* by one, and if *Counter* is nonzero, it sends a *New($F(D_k)$, Counter-1)* message

to all other of its neighbors. This way, the summary of the documents of the new node is propagated to the existing peers.

In addition, the new node must construct its own merged filters. This is achieved through a sequence of *FW(Filter, Counter, Flag)* messages. In particular, each node n_i upon receipt of a *New* message from a node n_j, it replies to n_j with a *FW(F(D_i), R, False)* message where *F(D_i)* is n_i's local filter and *Counter* is set to *R*. The use of the *Flag* parameter will be explained shortly. Upon receipt of a *FW(F(D_i), Counter, False)* message, each peer n_j, decrements *Counter* by one, and if *Counter* is nonzero, it sends a *FW(F(D_i), Counter-1, False)* message back to the peer that has sent the *New* message to it. This way, the local summaries reach the new peer n_k. Peer n_k creates its merged filters by merging the corresponding local filters received by the various *FW* messages.

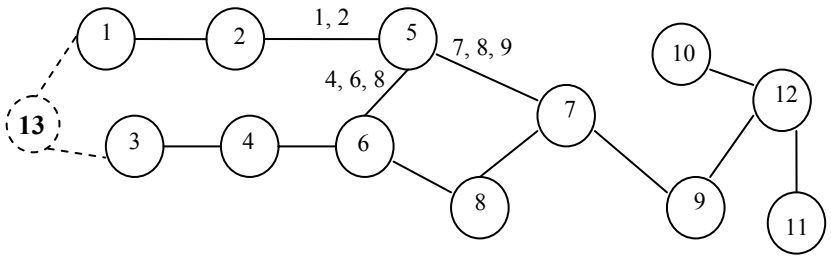

Fig. 2. Horizon-based distribution.

We now explain the use of the *Flag* parameter. *Flag* is used because the insertion of a new peer may change further the horizons of existing peers. Take for example the network of Figure 2 with $R = 2$. Say a new peer, peer 13, enters the network and links to both peers 1 and 3. The local filter of 13 must be propagated to 1, 2 and 3, 4; this is achieved through the *New* messages. Peer 13 must also construct its own merged filters; this is achieved through the *FW* messages with *Flag* equal to *False*. However, note that the insertion of 13 has changed the relative distance of some peers. In particular, now peer 3 (1) belongs to the horizon of 1 (3) since their distance (through the new peer 13) is now 2. Thus, the local filter of 3 (1) must now be merged with the corresponding filter of 1 (3).

Flag is used as follows. *Flag* is initially set to *False*. When the new node n_k receives a *FW(Filter, Counter, False)* message, it changes *Flag* to *True*, decrements *Counter* by one, and if *Counter* is nonzero, it propagates a message *FW(Filter, Counter-1, True)* to all of its other neighbors. Upon receipt of a *FW(Filter, Counter, True)* message, each peer merges the *Filter* with its corresponding merged filter, decrements *Counter* by one, and if *Counter* is nonzero, it sends a *FW(Filter, Counter-1, True)* message to its neighbors. This way, summaries of peers whose horizons change by the introduction of the new peer are propagated to each other.

When a peer wishes to leave the system, it sends an update message to all of its neighbours with a counter set to the radius *R*. When the message reaches a peer, the peer performs the update at its merged filter and propagates the message further until the counter reaches 0. Furthermore, it sends it own local filter through the same link

with a counter set to R to inform the peers that are now included in its horizon, since the departure of the peer has resulted in the decrease of its distance with other peers.

Note that, as indicated in Figure 2, it is possible that the local filter of a peer n_i is included in more than one merged filter of some other peer n_j. However, we may want to avoid this, because during search, two different paths will lead us to the same peer. This problem can be overcome by using peer identifiers. Each peer stores the identifiers of the peers that are included in each of its merged filters. When a local filter reaches a peer during the join procedure, the peer first checks whether it has already stored this filter at the merged filter of some other link.

2.5 Query Routing

We now discuss in detail how a query q posed at a peer n is processed. Our goal is to locate all peers that contain documents matching the query.

Peer n first checks it own local filter to see whether it matches the query. Then, it propagates the query only through one or more of those links whose merged filters match the query. Analogously, each peer that receives the query first check its own local filter and propagates the query only to links whose merged filters match the query. This procedure ends when a maximum number of peers has been visited or when the desired number of matching documents (results) has been attained.

When a query reaches a peer that has no link with a merged filter that matches the query, backtracking is used. This state can be reached either by a false positive or when we are interested in locating more than one matching document. In this case, the query is returned to the peer previously visited that checks whether there are any other links that match the query that have not been followed yet, and propagates it through one or more of them. If there are no such matching links, it sends the query to its previous peer and so on. Thus, each peer should store the peer that propagated the query to it. In addition, we may store an identifier for each query to avoid cycles.

We now describe a heuristic that can be used to choose which of the links that match a query to follow. The heuristic uses the counters of the matching merged filters to select the link through which we expect to find more matching documents. We describe first the idea for simple (single-level) Bloom filters.

Assume we have a query q with p element names: $\alpha_1, \alpha_2, \ldots, \alpha_p$. For each matching merged filter, we compute a value, called MIN, as follows. For each element α_i, the counters at the corresponding positions are checked and the minimum value $\min(\alpha_i) = \min(c(h_1(\alpha_i)),\ldots, c(h_k(\alpha_i)))$ is stored. Then, we take the overall minimum for all elements: $\text{MIN} = \min\{\min_i(\alpha_i)\}$, for $i = 1,\ldots, p$. This is the maximum number of results (matching documents) for q that can be found following the link with this merged filter. The query is propagated through the link whose merged filter has the largest value for MIN, because it is expected that the peers that can be reached through this link maintain the most results.

For multi-level filters, the procedure is slightly altered. For every element in the query, the counters of the corresponding level that gives a match are checked and the minimum value is selected. The minimum values of every element are added together ($\text{SUM} = \min(a_1) + .. + \min(a_p)$) and the link chosen is the link whose filter produces the largest such sum. If a path matches more than once in a single filter (the path

exists at different levels), the largest of the SUMs produced is chosen to be compared with the SUMs of the other filters.

3 Content-Based Overlay Networks

In this section, we discuss how the overlay network is created. The approaches refer to the way a peer chooses its neighbors in the overlay network when it joins the system.

3.1 Proximity-Based vs Content-Based Organization

We propose two approaches for organizing the peers. The first approach is based on network proximity, and the second one on filter similarity.

The *network proximity based* approach organizes the peers based on their proximity in the physical network. The motivation behind this organization is an effort to satisfy queries locally and minimize response time. Whenever a new peer n_i wants to join the system, it broadcasts a message to all the peers in the system. Peer n_i selects to be linked to those peers from which a response came first, since it is assumed that these are the closest ones. Using this organization it is expected that peers that are topologically close in the underlying physical network are going to be linked together in the overlay network.

The second approach organizes the peers based on *the similarity of their content*, that is, based on the similarity of their local documents. This approach attempts to group relevant peers together. The motivation for this organization is to minimize the number of irrelevant peers that are visited when processing a query.

Instead of checking the similarity of documents, we rely on the similarity of their filters. This is more cost effective, since a filter for a set of documents is much smaller than the documents themselves. Furthermore, the filter comparison operation is more efficient than a direct comparison between two sets of documents. Documents with similar filters are expected to match similar queries.

In this organization, a peer n_i that joins the system broadcasts its local filter $F(D_i)$. A peer n_j receiving the message replies with the distance of its own local filter from n_i's filter, $distance(F(D_i), F(D_j))$. Peer n_i chooses to attach to the peers that returned the smallest distances, i.e., the most similar ones.

This way peers with relevant content are expected to be grouped together so as to form content-based clusters of peers. With this organization, once a query enters the relevant cluster, peers with matching documents are expected to be within reach.

3.2 Filter-Based Similarity Metric

For Bloom-based filters the distance function used to evaluate the degree of similarity between two filters BF_1 and BF_2, $distance(BF_1, BF_2)$, is computed using the Hamming distance. This distance corresponds to the number of bits at which the two Bloom filters differ. The more similar the two documents are, the smaller their Hamming distance is. An example is shown in Figure 3(a). For multi-level Blooms, to

compute their distance, the distance of each level is calculated the same way as for a simple Bloom, and the results are added together.

Figure 3(b) illustrates an experiment that confirms the validity of the metric. We used different percentage of elements repetition between documents and measured their distance. The distance decreases linearly with the increase of the repetition between the documents. The same holds for the similarity between multi-level Blooms, although in this case, the metric depends on the structure of the documents as well.

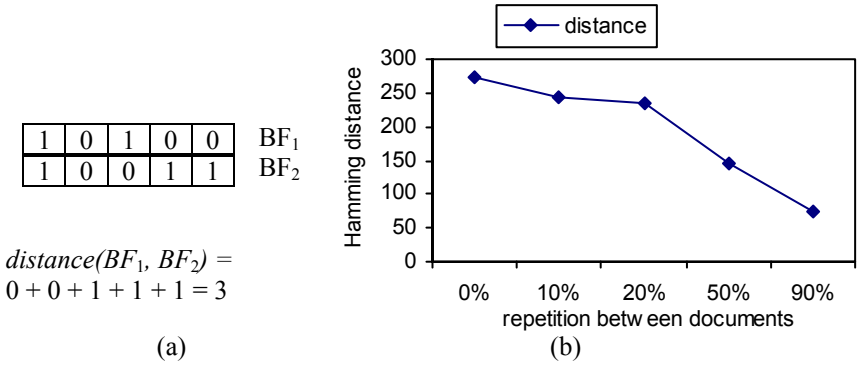

Fig. 3. (a) Similarity metric (b) Document and filter similarity.

4 Performance Evaluation

4.1 Simulation Model

We simulated the peer-to-peer network as a graph where each node corresponds to a peer. A number of XML documents are associated with each node. We simulated the organization of the peers using horizons both with and without the use of filters and compared both the proximity and the content-based organizations. For the first two experiments we used simple Bloom filters as summaries and queries of length 1. In the last two experiments, we show how multi-level filters outperform simple ones in this distributed setting when evaluating queries with length greater than 1. We used only Breadth Bloom filters in our experiments; Depth Bloom filters are expected to perform similarly with Breadth Bloom filters [9].

For the hash functions, we used MD5 [13] that is a cryptographic message digest algorithm that hashes arbitrarily length strings to 128 bits. The k hash functions are built by first calculating the MD5 signature of the input string, which yields 128 bits, and then taking k groups of $128/k$ bits from it. We select MD5 because of its well-known properties and relatively fast implementation. For the generation of the XML documents, we used the Niagara generator [14] that generates tree-structured XML documents of arbitrary complexity. It allows the user to specify a wide range of characteristics for the generated data by varying a number of input parameters that control the structure of the documents and the repetition between the element names.

The structure of the network depends on: the diameter of the graph (the maximum distance between any two nodes through the shortest available path), the number of nodes (peers), the number of edges (links), the maximum fanout for a peer and the radius of the horizon. In our experiments, the size of the network was set to 100 peers and its diameter to 30 hops, the filter size was fixed to 2000bits with 4 hash functions. XML documents have 5 levels and 80 elements and different structure. Every 10% of the documents are 50% similar to each other in terms of element names. Query routing stops when a maximum number of hops is traversed. The maximum number of hops should be large enough to cover a sufficient proportion of the network without producing excessive delays to a query. Our performance metrics are the percentage of successful queries (i.e., queries that found at least one matching node) and recall (i.e., the percentage of results found).

Table 1. Summary of performance parameters.

Parameter	Default Value	Range
Filter size	2000 bits	
Number of hash functions	4	
Number of queries	200	
Number of elements per document	80	
Number of levels per document	5	
Length of query	1	1-3
Number of nodes	100	
Percentage of hits (matching nodes per query) in the network	7%	
Radius of horizon	5	3-11
Diameter	30	
Max fanout	5	
Max number of hops	30	
Number of hits per node	1	1-50

4.2 Experimental Results

Experiment 1: Radius (R)

At this first experiment, we examine the influence of the radius. The network structure was fixed to 100 nodes with diameter 30 and we varied the radius from 3 to 11. We measured the percentage of successful queries and the percentage of results found (recall), with and without the use of Bloom Filters.

Bloom filters improve the performance of search for any value of the Radius (*R*). Radius 3 gives the smallest number of results and the smallest percentage of successful queries. This is expected because search for matching documents is limited to peers within distance 3 from the peer issuing the query, thus matching peers furthest away are never visited. Radius 5 gives the most results and the best percentage of successful queries. Increasing the Radius further does not improve performance. There are two reasons for this. First, as the radius increases, the number of peers that correspond to each merged Bloom filter also increases. This leads to more false positives for a same size filter. Second, when the radius becomes relatively large, the merged filter of each link summarizes the content of a very large number of

peers reached through this link. Thus, a path may be followed to a matching peer located very far away from the peer issuing the query instead of a path to a near-by matching peer. In the rest of our experiments we set $R = 5$.

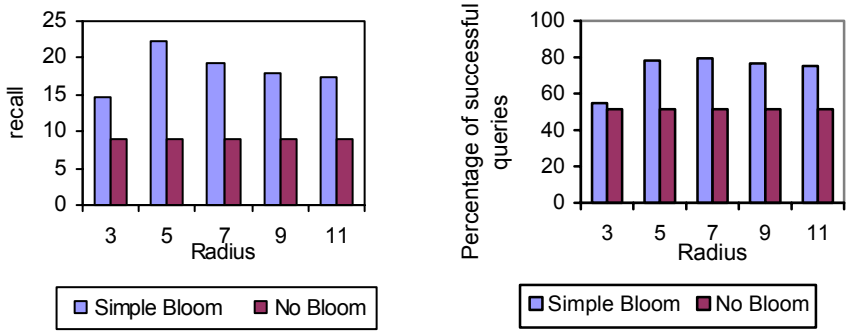

Fig. 4. Influence of the radius.

Experiment 2: Proximity-Based vs Content-Based

At the second set of experiments, we study content-based distribution. The performance of content-based distribution depends on whether, for a given query, we are able to locate the cluster with peers that have results similar to it. Once in the right cluster, we are able to find all results nearby, since the matching peers are linked together. To model this, we varied the percentage of queries issued from a peer that matches the query from 0 to 100% (that is, the percentage of queries that start from the correct cluster). As shown in Figure 5(left), once in the right cluster (100% query distribution), we are able to locate all matching documents, whereas with the proximity-based distribution less than 30% of the matching nodes are identified. On the other hand, proximity-based organization has a larger percentage of successful queries, since the matching peers for each query are distributed randomly across the network. On the contrary, in the content-based organization, all peers with similar documents are clustered together. Thus, if the requesting peer is far from the correct cluster, it may not be able to find any matching peer in its cluster.

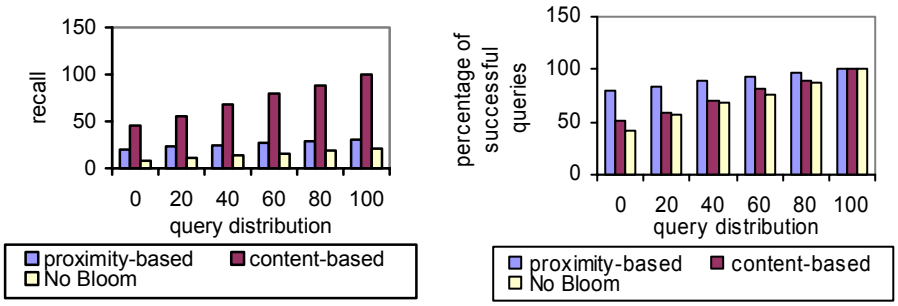

Fig. 5. Varying query distribution.

Figure 6 shows the effectiveness of our content-based join procedure. It depicts for a query and a matching node n, the percentage of matching nodes within distance

R from *n*. For *R* larger than 5, all matching nodes are within the horizon, that is, they are within this distance and thus they can be located efficiently.

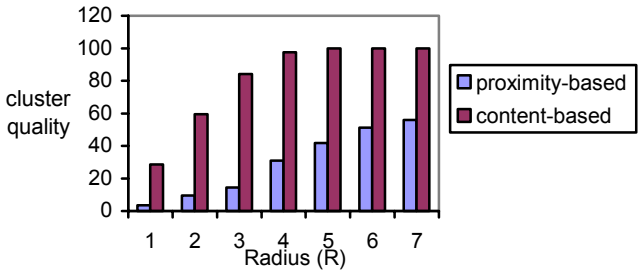

Fig. 6. Quality of clustering (percentage of matching nodes with respect to R).

Experiment 3: Breadth vs Simple Bloom Filters

In this set of experiments, we used Simple and Breadth Bloom filters to demonstrate that our distribution mechanisms can also be used with multi-level filters. Our queries in this example are path queries of length 3. We denote with *P* the percentage of documents that contain the element names that appear in the path query but without them forming the actual path. In the first experiment (Figure 7 (left)), we vary the radius *R* setting P = 75%, while in the second experiment (Figure 7(right)), we vary *P* setting *R* = 5. The results of the first experiment for the multi-level are analogous to those of Experiment 1 (Figure 4(left)), but the percentage of results for the simple Bloom Filter is nearly constant. This happens because the number of false positives is very large for any value of the radius for path queries. Figure 7(right) shows that for *P* = 0 both filters perform nearly the same, but as *P* increases the performance of the simple Bloom Filter reduces due to the increase of false positives.

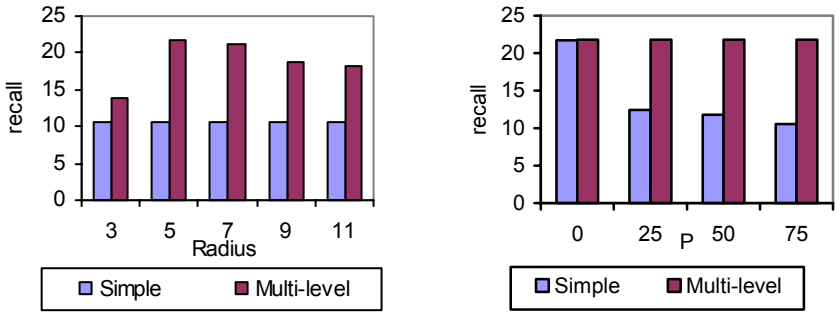

Fig. 7. Simple and multi-level Bloom filters.

Experiment 4: Use of Counters

At this last experiment, we examine how the use of the counters heuristic improves performance, both for Simple and Breadth Blooms. We varied the number of hits (matching documents) per peer and measured the percentage of results found and the percentage of successful queries. The variance of the results is a value that shows the average difference between the number of hits to each peer (that maintains results) and the average number of hits of all that peers at square power. More specifically,

variance = $\sum_{i=1,..,k}(x_i-\mu)^2/k$, where k is the number of peers that have results, x_i is the number of hits for peer n_i and $\mu = (\sum_{i=1,..,k} x_i) / k$ (average hits per peer that has results).

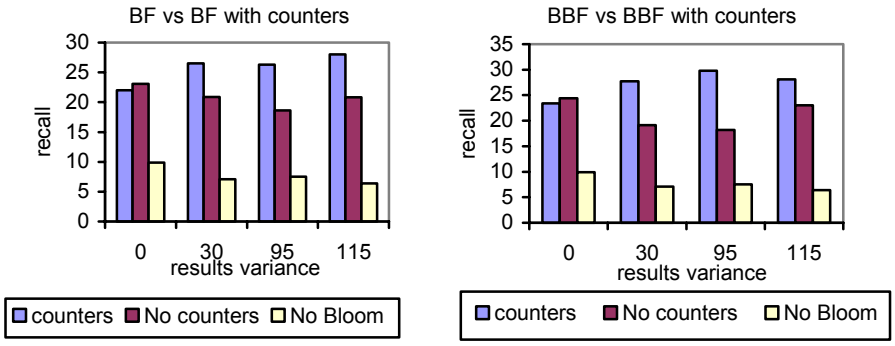

Fig. 8. Use of counters.

Figure 8 shows that in both cases (simple (BF) and multi-level (BBF) Bloom filters) for zero variance (which means that all peers have the same number of results) using or not counters results in the same performance. For non-zero variance, the use of counters gives better performance since the query is propagated to links through which we expect more results to be found.

5 Related Work

In the context of peer-to-peer computing, many methods for resource discovery have been proposed. These methods construct indexes that store summaries of other nodes and additionally provide routing protocols to propagate the query to the relevant nodes.

Bloom filters have been used as summaries in such a context. The resource discovery protocol in [4] uses simple Bloom filters as summaries. Servers are organized into a hierarchy modified based on the query workload to achieve load balance. Each server stores summaries, which are used for query routing. Summaries are a single filter with all the subset hashes of the XML documents up to a certain threshold. To evaluate a query, the query is split to all possible subsets and each one is checked in the index. Another method based on Bloom filters for routing queries in peer-to-peer networks is presented in [7]. It is based on a new structure, called attenuated Bloom filter, residing in every node of the system. The filter stores information about nodes within a range of the local node and uses a probabilistic algorithm in order to direct a query. The algorithm either finds results quickly, or fails quickly and exhaustive searching is then deployed. Another use of Bloom filters as routing mechanisms is proposed in [11]. Local and Merged Blooms are used, but there is no horizon to limit the information a node stores about other nodes, and thus scalability is an issue. Also the filters are constructed based on file names and not on file content. A similar approach is followed in [6], where routing indices (other than

Blooms), placed at each node, are used for efficient routing of queries. By keeping such an index for each outgoing edge, a node can choose the best neighbor for forwarding the query. The choice is based on summarized information about the documents along that path, which is stored in the index.

However, all these methods do not provide any grouping of the nodes according to their content, and use summaries only for routing and not for building the overlay network. Also they are limited in answering membership queries and not path queries.

Content-based distribution was recently proposed in [8] which introduced Semantic Overlay Networks (SONs) [8]. With SONs, nodes with semantically similar content are "clustered" together, based on a classification hierarchy of their documents. Queries are processed by identifying which SONs are better suited to answer it. However, there is no description of how queries are routed or how the clusters are created and no use of filter or indexes to efficiently locate the node that stores a particular data item. Schema-based peer-to-peer networks provide an approach that supports more complex metadata clustering than previous work and thus can support more complex queries. An RDF-based peer-to-peer network is presented in [15]. The system can support heterogeneous metadata schemes and ontologies, but it requires a strict topology with hypercubes and the use of super-peers, limiting the dynamic nature of the network. In [16], attribute-based communities are introduced. Each peer is described by attributes representing its interests; the emphasis is on the formation and discovery of communities. Since the communities are attribute-based they are less expressive than schema-based or content-based networks and support less complex queries. In [10], documents are classified into categories based on keywords and metadata. Nodes are then clustered based on these categories. Focus is given on load-balancing.

Chord [12] is a representative of structured p2p networks that uses a distributed lookup protocol designed so that documents can be found with a very small number of messages. It maps keys and nodes together to improve search efficiency. However, this approach lacks node autonomy and provides no grouping between nodes with similar content.

Our work relates also to distributed processing of XML queries. Recent work in this context includes [18], where a cost-model for the distribution of XML documents and a query decomposition technique are presented for querying distributed and (partially) replicated XML data.

6 Conclusions

In this paper, we have presented multi-level Bloom filters that are hash-based indexing structures that can be used for the representation of hierarchical data and support the evaluation of path queries. We showed how such filters can be distributed using a horizon-based organization to support the efficient routing of queries in a p2p network. In addition, we described how content-based overlay networks can be built using a procedure that clusters together peers with similar content. Similarity of peer content is based on the similarity of their filters. Our performance results show that content-based overlay networks built by this procedure are very efficient in locating a large number of peers with matching documents.

Future work includes dynamic updates of "clusters" when the content of peers is modified. In addition, we plan to experiment with p2p configurations with properties that match those of popular p2p networks as indicated by current measurement studies (e.g. [19], [20]).

References

1. B. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM, 13(7), July 1970.
2. L. Fan, P. Cao, J. Almeida, A. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. In Procs of ACM SIGCOMM Conference, Sept. 1998.
3. S.D. Gribble, E.A. Brewer, J.M. Hellerstein, D. Culler. Scalable Distributed Data Structures for Internet Service Construction. In Procs of the 4th Symposium on Operating Systems Design and Implementation, 2000.
4. T.D. Hodes, S.E. Czerwinski, B.Y. Zhao, A.D. Joseph, R.H. Katz. Architecture for Secure Wide-Area Service Discovery. Mobicom '99.
5. M.V. Ramakrishna. Practical Performance of Bloom Filters and Parallel Free-Text Searching. Communications of the ACM, 32 (10), October 1989
6. A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-peer Systems. In ICDCS, 2002.
7. S. C. Rhea, J. Kubiatowicz. Probabilistic Location and Routing. In INFOCOM, 2002.
8. A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Submitted for publication.
9. G. Koloniari and E. Pitoura. Bloom-Based Filters for Hierarchical Data. 5th Workshop on Distributed Data Structures and Algorithms (WDAS), 2003.
10. P. Triantafillou, C. Xiruhaki, M. Koubarakis, N. Ntarmos. Towards High Performance Peer-to-Peer Content and Resource Sharing Systems. CIDR 2003.
11. A. Mohan and V. Kalogeraki. Speculative Routing and Update Propagation: A Kundali Centric Approach. IEEE International Conference on Communications (ICC'03), May 2003.
12. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. Procs. of the 2001 ACM SIGCOMM Conference.
13. The MD5 Message-Digest Algorithm. RFC1321.
14. The Niagara Generator, http://www.cs.wisc.edu/niagara
15. W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, A. Loser. Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks. WWW 2003, May 2003, Budapest, Hungary. ACM 1-58113-680-3/03/0005.
16. M. Khambatti, K. Ryu, P. Dasgupta. Peer-to-Peer Communities: Formation and Discovery. 14th IASTED International Conference on Parallel and Distributed Computing and Systems, 2002.
17. D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing, HP Technical Report, HPL-2002-57
18. S. Abiteboul, A. Bonifati, G. Cobéna, I. Manolescu, T. Milo. Dynamic XML Documents with Distribution and Replication. SIGMOD 2003.
19. S. Saroiu, K. Gummadi and S. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. Proceedings of Multimedia Computing and Networking (MMCN), 2002.
20. F. S. Annexstein, K. A. Berman and M. A. Jovanovic. Latency Effects on Reachability in Large-scale Peer-to-Peer Networks. Procs of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures, 2001.