Distributed Pattern Matching for P2P Systems

Reaz Ahmed
School of Computer Science
University of Waterloo
Ontario, Canada N2L 3G1
Email: r5ahmed@uwaterloo.ca
Telephone: (519) 888–4567 ext.4716

Raouf Boutaba
School of Computer Science
University of Waterloo
Ontario, Canada N2L 3G1
Email: rboutaba@bbcr.uwaterloo.ca
Telephone: (519) 888–4820
Fax: (519) 885–1208

***Abstract*— Flexibility and efficiency are the prime requirements for any P2P search mechanism. Existing P2P systems do not seem to provide satisfactory solution for achieving these two conflicting goals. Unstructured search protocols (as adopted in Gnutella and FastTrack), provide search flexibility but exhibit poor performance characteristics. Structured search techniques (mostly Distributed Hash Table (DHT)-based), on the other hand, can efficiently route queries to target peers but support exact-match queries only. In this paper we present a novel P2P system, called Distributed Pattern Matching System (DPMS), for enabling flexible and efficient search.**

Distributed pattern matching can be used to solve problems like wildcard searching (for file-sharing P2P systems), partial service description matching (for service discovery systems) etc. DPMS uses a hierarchy of indexing peers for disseminating advertised patterns. Patterns are aggregated and replicated at each level along the hierarchy. Replication improves availability and resilience to peer failure, and aggregation reduces storage overhead. An advertised pattern can be discovered using any subset of its 1-bits; this allows inexact matching and queries in conjunctive normal form. Search complexity (i.e., the number of peers to be probed) in DPMS is $O\left(\log N + \xi \log \frac{N}{\log N}\right)$, where N is the total number of peers and ξ is proportional to the number of matches, required in a search result. The impact of churn problem is less severe in DPMS than DHT-based systems. Moreover, DPMS provides guarantee on search completeness for moderately stable networks. We demonstrate the effectiveness of DPMS using mathematical analysis and simulation results.

I. Introduction

The generic pattern matching problem and its variants have extensively been studied in Computer Science literature. In this paper we focus on a variation of the pattern matching problem that conforms to two constraints. First, we consider Bloom-filter based pattern matching with don't care bits, and second we assume that the patterns are scattered among the peers of a P2P overlay network (see Fig. 1). Each pattern summarizes the properties of a shared object (such as a file or a service) and is a couple of hundred bits long. One possible form of such a pattern is a bloom-filter [5] obtained from the properties of an object. Another possibility is to use some predefined encoding of object properties, as adopted in figureprint construction techniques in molecular biology.

A peer can initiate a query by using a pattern. If the query pattern is exactly the same as the advertised pattern then the problem can efficiently be solved using conventional DHT techniques. But, we are interested in inexact pattern matching,

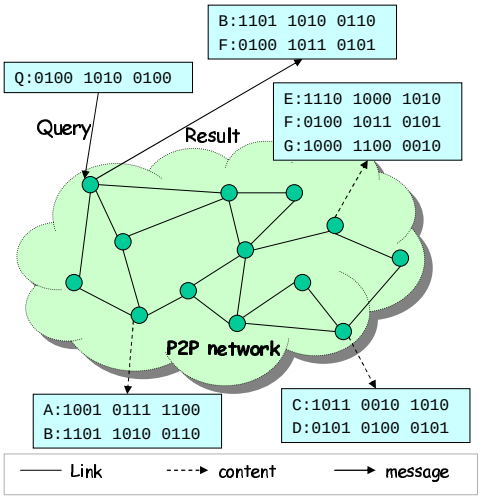


Fig. 1. **Distributed Pattern Matching**

where the 1-bits of a query pattern can be any subset of an advertised pattern that it should match against. In this paper we present a new P2P architecture for solving this Distributed Pattern Matching (DPM) problem and demonstrate the effectiveness of the proposed architecture using mathematical analysis and simulation results.

Keyword searching is one of the essential functionalities offered by any peer-to-peer file-sharing system. A Centralized filesystem, as present in any traditional operating system, permits more sophisticated search operations involving wildcards and partial keywords. Enabling existing P2P file-sharing systems with wildcard search capability will allow users to perform more flexible and powerful searches. Besides inexact keyword matching, many problems like partial service description matching for service discovery systems, data record pre-scanning for distributed database systems, molecular fingerprint matching in a distributed environment, etc., can be mapped to and efficiently solved using Distributed Pattern Matching (DPM).

Over the last few years DHT-based structured P2P systems ([19], [16], [15], [23] etc.) have gained importance due to efficiency of routing queries and search completeness. Most of these techniques take binary keys as input and apply prefix matching to route a query to a specific node in $O(\log N)$ hops,

where N is the number of peers in the overlay. Despite their efficiency in query routing, these systems are not suitable for solving DPM problem. The basic idea behind DHT-techniques is to partition the key-space into non-overlapping regions and to assign each region to a peer bearing an ID from that region. But from pattern matching perspective it is quite difficult to partition even one dimensional pattern (or key) space into non-overlapping clusters, while preserving the notion of closeness in patterns.

Unstructured search techniques, like flooding [2] and random walk [14], can be used to solve DPM problem. But the generated traffic for searching is proportional to the number of peers in the overlay and there is no guarantee on search completeness. Hence adopting a unstructured search technique is not a good choice for solving DMP problem.

In this paper we present a P2P system, DPMS (Distributed Partial Matching System), for efficiently solving the DPM problem. DPMS uses replication and aggregation for distributing patterns advertised by peers across the P2P overlay. In DPMS, peers collaborate to form a lattice-like indexing hierarchy. This hierarchy is used to efficiently route queries to target peer(s).

DPMS has several properties of an unstructured P2P system. First, it supports flexible queries involving partial and multiple keywords. Second, placement of documents is not controlled by the system. Third, it can exploit the heterogeneity in peer capabilities. Unreliable and less capable peers contribute to less important parts of the indexing hierarchy, while reliable (long lived) and powerful (in terms of storage and connection speed) peers take responsibility of more important parts of the indexing hierarchy.

To avoid flooding and to provide an efficient query routing mechanism, DPMS uses the indexing hierarchy. The philosophy behind this architectural choice is that, by building an indexing hierarchy of height $O(\log \frac{N}{\log N})$ we can have $O(\log N)$ peers at the highest level. Peers at any level collectively covers all the leaf peers (hence all the advertised patterns) residing at the bottom level of the indexing hierarchy. In other words, we can check all the patterns at the bottom level by probing only $O(\log N)$ peers at level $O(\log \frac{N}{\log N})$. This means we can find κ leaf peers containing match for a given query pattern in $O(\log N + \varepsilon\kappa \log \frac{N}{\log N})$ probes. The $O(\log N)$ probes is the cost of flooding at the topmost level and $O(\varepsilon\kappa \log \frac{N}{\log N})$ is the cost of reaching the κ matching leaf peers along the indexing hierarchy of height $O(\log \frac{N}{\log N})$. The term ε depends on the amount of false positives introduced by the lossy aggregation scheme.

In such a hierarchy the topmost level peers will receive a very high volume of queries and will become performance bottleneck. Besides, fault-tolerance characteristics of the system will be poor; failure of any peer along the indexing hierarchy will result into unreachable leaf peers. To overcome these problems DPMS uses replication at each level of the indexing hierarchy.

With such a replication strategy, the network and storage overhead for index maintenance will be high. To reduce the impact of replication overheads, we have incorporated a *don't care*-based lossy aggregation scheme at each level of the indexing hierarchy. The proposed aggregation scheme allows incorporation of multiple advertised patterns or aggregates in a single aggregate, while making it possible to perform matching of a query pattern against the aggregates. The aggregation scheme increases the chances of false positives while routing queries. Simulation results indicate that we can achieve $45 - 60\%$ reduction in storage and network overhead while securing query routing efficiency almost identical to the ideal case, i.e. without aggregation (see. Fig 5).

To our knowledge, Distributed Pattern Matching (DPM) problem has not been addressed by any research activity in the peer-to-peer context, so far. The index distribution architecture of DPMS is unique and has been designed to specifically solve the DPM problem. The novel aggregation scheme, proposed in this paper, can effectively reduce storage overhead at the indexing peers without incurring a significant decrease in query routing performance. However, the use of bloom filter for representing indices is not new. Many network applications use bloom filters. A comprehensive list of such applications can be found in [6].

The rest of this paper is organized as follows. The architecture and operation of DPMS are presented in section II. Mathematical analysis of search complexity in DPMS is provided in section III. Section IV presents experimental results. Section V provides a description and brief comparison of the existing approaches with DPMS. Finally, concluding remarks are presented in section VI.

II. The new system

This section presents details on DPMS architecture. In this section we will use the terms *pattern* and *index* interchangeably, as patterns are used as indices for query routing.

A. Overview

In DPMS a peer can act as a leaf peer or indexing peer or both. A leaf peer is at the bottom level of the indexing hierarchy and advertises its indices (created from the objects it is willing to share) to other peers in the system. An indexing peer, on the other hand, stores indices from other peers (leaf peers or indexing peers). A peer can join different levels of the indexing hierarchy and can simultaneously act in both roles. Indexing peers get arranged into a lattice-like hierarchy (see Fig. 2) and disseminate index information using repeated aggregation and replication.

Index (e.g. keywords or hash keys) replication is used by many unstructured P2P systems for improving reliability and availability. But replication incurs extra overhead on storage and network bandwidth. To improve efficiency, these systems adopt smart replication strategies [8].

DPMS uses replication trees (see Fig. 2a) for disseminating patterns from a leaf peer to a large number of indexing peers. However such a replication strategy will generate a large volume of traffic, and is not feasible for any practical implementation. To overcome this shortcoming, DPMS combines

replication with aggregation to minimize the volume of traffic between peers in adjacent levels in the indexing hierarchy. As shown in Fig. 2b, advertisements from different peers are aggregated and propagated to peers in the next level along the aggregation tree.

The amount of replication and aggregation is controlled by two system-wide parameters, namely replication factor R and branching factor B. In order to achieve constant volume of messages exchanged between adjacent levels, an aggregation ratio of $R : 1$ is required.

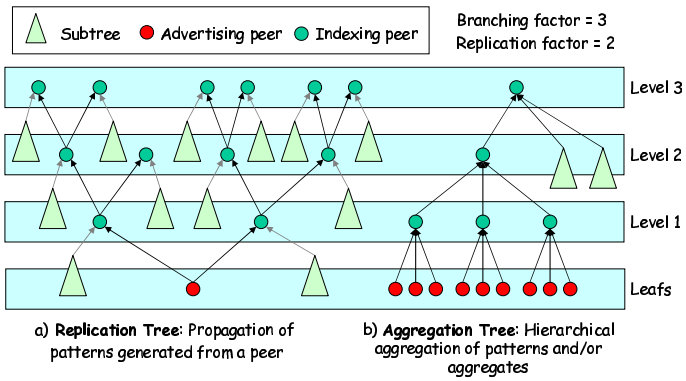


Fig. 2. **DPMS overview**

Patterns advertised by a leaf peer are propagated to R^l indexing peers at level l. On the other hand, an indexing peer at level l contains patterns from B^l leaf peers. Due to repeated aggregation, the aggregates become more generic (i.e. lower information content) as we move up along the indexing hierarchy.

The indexing hierarchy has three-fold impact on system performance. Firstly, the indexing hierarchy evenly distributes index information (and queries) in highest level indexing peers. This helps in load balancing the system and improves fault tolerance. Secondly, peers can route queries towards target leaf peer(s) without having any global knowledge of the overlay topology. Each peer needs to know the addresses of its children, replicas and one of its parents. Finally, the indexing hierarchy helps in minimizing query forwarding traffic. While forwarding a query from a root peer to multiple leaf peers in the same aggregation tree, shared path from the root peer to the common ancestor of the target leaf peers is utilized.

B. Index/pattern construction from keywords

In DHT-based systems an index is obtained by applying some system-wide known hash function to the keyword(s) related to a document. In DHT-techniques an index is used in two ways. First, to identify a document, and second, to identify the peer responsible for that document (or a pointer to that document). A query consists of an index, created by hashing the search keyword. This warrants the search index to be identical to the advertised index. However, a peer can readily identify the responsible peer for a query, and route the query to that peer efficiently.

In unstructured systems, on the other hand, documents are identified using associated keywords. A query consists of one or more search keywords. Query routing is done based on flooding or random walk. A peer receiving a query can return a document partially matching the search keyword(s), in case an exact match was not found.

DPMS uses Bloom filters [5] as indices, to achieve the advantages of both unstructured and structured P2P systems, i.e. efficient routing and inexact matching.

Bloom filters are used to test set membership. Because of their space-efficiency, Bloom filters are used in many network applications for exchanging content summary between networked nodes [6]. However, this space-efficiency comes at the expense of a small possibility of false positives in the membership check operation.

The algorithm for Bloom filter construction is simple. A Bloom filter is represented as a m-bits array. k different hash functions are also required to be defined. Each of these hash functions should return values within the range of $[0..m)$. In an empty Bloom filter all the m-bits are set to 0. To insert an element (a string or keyword), it is hashed with the k hash functions and corresponding k array positions are set to 1. To test set membership for an element, it is hashed with the k hash functions to get k array positions. If all of these k-bits are set (i.e. 1), then with high probability the element is a member of the set represented by the Bloom filter, otherwise it is not.

Each document in a traditional file-sharing P2P system is associated with a set of keywords. In DPMS, all the keywords associated with a document are encoded in a single Bloom-filter. To facilitate inexact matching, each keyword is first fragmented into n-grams (usually trigrams). These n-grams are then inserted into the Bloom filter representing the document.

Query keywords are also fragmented into n-grams (see Fig. 3) and encoded into a Bloom filter. The 1-bits on a query should be subset of any pattern that it should match against. This kind of encoding allows us to retrieve documents, advertised with keywords *"invisible man"* and *"visible woman"* respectively, using a query containing partial keywords like *"visi man"*.

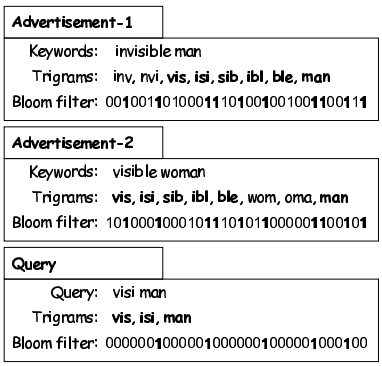


Fig. 3. **Bloom filter example of two advertisements and an inexact multi-keyword query. 1-bits in boldface corresponds to the matching trigrams.**

For a P2P service discovery system the index can be obtained in a similar fashion using attribute-value pairs instead of keywords. Molecular fingerprint can be used as index for some envisioned distributed system storing molecular structure information.

C. Aggregates

DPMS relies on replication for disseminating pattern information along replication trees. Replication is necessary for load-balancing and for improving fault-tolerance. The replication strategy adopted in DPMS would significantly increase network and storage overhead. DPMS uses repeated aggregation at each level along the hierarchy to mitigate this problem.

We suggest a don't care based aggregation scheme, i.e. don't cares (presented by X) are used to represent both 1 and 0 in the same bit position. Don't cares are used at the positions where the constituent patterns disagree.

This type of aggregates retains parts from the constituent patterns or aggregates. A 1-bit (or 0-bit) in such an aggregate indicate that all the patterns contributing to this aggregate had 1 (or 0) at corresponding position. However incorporating this extra information (i.e. X's) incur some space overhead, which can be minimized by compressing the aggregates using huffman encoding or run length encoding during transmission through the network.

An indexing peer acts as a multiplexer in the indexing hierarchy. It gathers *in-list*s (lists of patterns or aggregates from the B child peers), aggregates them to another list (referred to as *out-list*) of aggregates, and sends this list to each of its parents.

Construction of this out-list is not trivial. We want the aggregates in the out-list to have a minimum number of X-bits. This ensures minimum information loss. The problem of obtaining an out-list containing minimal number of X-bits is NP-complete. Instead we use a heuristic approach to measure the degree of similarity between two patterns/aggregates. Experimental results show that a near optimal out-list can be produced by combining patterns/aggregates with highest degree of similarity.

As a measure of the degree of similarity between two patterns/aggeregates, we have used the percentage of positions at which they agree. To compute out-list from the B in-lists, obtained from B children, we have used an iterative algorithm. At each iteration step, the pair of patterns with highest degree of similarity were combined and the resultant aggregate was inserted into the (intermediate) out-list for consideration in the next iteration step.

D. Index Distribution

An indexing peer, participating in the DPMS architecture, belongs to two sets, a vertical set (i.e., level) and a horizontal set (i.e., group). According to the degree of aggregation, each indexing peer belongs to a level (vertical set). Peers participating at higher index levels cover (i.e., contain index information from a) higher number of leaf peers along the aggregation tree. But, due to increased level of aggregation the contained information gets vaguer at higher levels.

Indexing peers at level l arrange into R^l groups (horizontal sets), numbered from 0 to $(R^l - 1)$ (see Fig. 4). In the ideal case, all the indexing peers in a single group (at any level) should collectively cover all the leaf peers in the system.

A peer at level l and group g $(0 \leq g < R^l)$ is responsible for transmitting its aggregate information to R parents at level $(l + 1)$. Each parent belongs to a different group, in range $[g \times R, (g + 1) \times R)$, respectively.

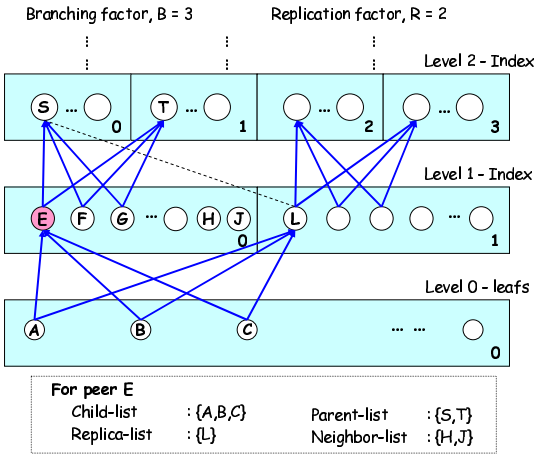


Fig. 4. **Index distribution architecture**. All the peers interacting with peer E are labelled. Group number is printed at the bottom right corner of each box.

Peers at level l and group g organize into subgroups (referred to as siblings) of size B to forward their aggregated information to the same set of parents. Thus each group in range $[g \times R, (g + 1) \times R)$ at level $(l + 1)$ will contain a peer replicating the same index information. This provides redundant paths for query forwarding and increases tolerance to peer failure.

E. Topology maintenance

In the DPMS index distribution hierarchy peers interact with each other in different roles, e.g. parent, child, neighbor etc. An indexing peer, say E belonging to level l and group g, maintains four separate lists for this purpose (see Fig. 4).

1) **Replica-list** contains the list of peers in the adjacent groups that have common children as that of E. This list contains $(R - 1)$ peers, one from each group in range $[\lfloor g/R \rfloor \times R, (\lfloor g/R \rfloor + 1) \times R)$, excluding g.
2) **Parent-list** This is the replica list obtained from one of E's parents. E uses this list to forward its aggregate information (out-list) to all of its parents along a replication tree.
3) **Child-list** contains the list of all children and the replica list for each of them. A peer normally communicates with the child peers only. But in case of a failure of a child it can communicate with a replica of the failed child. This list contains B entries corresponding to the B children of E at level $(l - 1)$ and group g/R.

4) **Neighbor-list** contains a fixed number of non-sibling peers that are in the same group (g) as peer E. This list is mostly used for maintaining connectivity in a group, during join operation, and for flooding queries horizontally within a group (mostly at the topmost group).

Out of these four lists a peer needs to keep track of three: child-list, parent-list and neighbor-list. The replica-list of a peer is the parent-list of any of its children. Peers use the Newscast protocol [21] for maintaining and updating these lists, i.e., to detect peer failures and arrival of new peers. Flow of news packets is restricted to $2 \times R$ groups of peers. More specifically, the news-list of a peer at level l and group g will contain information about some peers from groups $[\lfloor g/R \rfloor \times R, (\lfloor g/R \rfloor + 1) \times R)$ at level l and groups $[g \times R, (g+1) \times R)$ at level $(l+1)$. That is, each peer sends news packets to its parents, neighbors and replicas, and receives news packets from its children, neighbors and replicas.

Unlike indexing peers a leaf peer maintains only the neighbor-list and forwards this list to its parents. A leaf peer obtains its parent-list from one of its parents. It should be noted that leafs peers do not have any replica-list or child-list.

F. Query Routing

A query can be initiated by any peer in the system. The query life-cycle can be divided into three phases: ascending phase, blind search phase and descending phase.

During the *ascending phase*, an initiating (or intermediate) peer, checks its local information for the existence of a match. If a match is found, then the query is forwarded to the matching child, otherwise it is forwarded to any of its parents. This precess recurs until the query hits a peer with a match, or reaches a highest level peer.

The *Blind search phase* is executed by a highest level peer, say Z, upon receiving a query (from a child) that does not match any aggregate in its aggregate-lists. Z floods the query to all other peers in its group. If no peer in a group at the highest level contains a match, then the query was for a non-existent pattern, and so the search fails.

A query enters into the *descending phase* when it hits a peer containing a matching aggregate. The query is then forwarded to the child peer advertising the matching aggregate. This process recurs until the query reaches a leaf peer. Two types of exceptions may occur. Firstly, a false match may occur and the search branch terminates. Secondly, a peer may have multiple children matching the query and multiple search branches can be initiated. Priority and the order in which search branches are initiated is guided by predefined policy and application-specific requirements.

G. Node Join

A peer can join the system as a leaf peer or an indexing peer or both. To join as a leaf, a peer say C, has to find a level 1 indexing peer, say P, with an empty slot in its child-list. C joins the indexing hierarchy as a child of P. C obtains the replica-list from P and set these values and P as its parent-list. Then C can start advertising its patterns to all the peers in its parent-list. If C fails to find a level 1 peer with an empty slot, then it can either join in both level 1 and level 0 or select a level 1 peer with lowest number of children.

To join the indexing hierarchy as an indexing peer, a peer (say E) has to go through the following steps:

- **Choose level and group**: Peer E has to choose a level, say l in the hierarchy. Selection of level can be based on the nodes capacity, uptime distribution etc. Peers with higher capacity (storage and bandwidth) and longer life-time are expected to join higher levels in the indexing hierarchy. Then peer E can choose a group g in random such that g is in $[0, R^l)$.
- **Construct child-list**: Joining Peer E has to contact a seed peer to get information about other peers in the system. Peer E can crawl the index hierarchy to reach a peer, say A, such that peer A is in level $(l-1)$ and in group $\lfloor g/R \rfloor$, and the parent-list of peer A contains less than R entries. Peer E can join as a parent of peer A. Peer E has to join the group in which peer A has no parents. Peer E has to obtain and update the replica-list of other parents of peer A. Peer E can obtain the child-list from a parent of peer A. Peer A can have an empty parent-list during the initialization phase of the system or after a failure of all of its parents. If peer A returns an empty parent-list, then peer E should look for other (upto B) peers, in the same group as that of peer A, with empty parent-list. If such peers exist then peer E should make them its children.
- **Construct parent-list**: To construct the parent-list peer E has to find a peer, say T, such that peer T is in level $(l+1)$ and in group $(g \times R)$, and T has an empty slot in its child-list. If such a peer (T) exists then peer E constructs its parent list using peer T and all the replicas of peer T. Otherwise, peer E will start with an empty parent-list, and will wait for more peers to join at level $(l+1)$.

H. Node Leave or Failure

In DPMS, peer departure and failure are handled in the same manner, i.e., a peer can leave the system without any notice. The absence of an indexing peer, say E, will affect the peers in its parent-list, child-list and replica-list. Parents and children of E can still communicate through any of the replicas of peer E. So query routing is not hampered until all of the replicas of a peer fail.

Failure or departure of a leaf peer has greater impact on the system. All the index information along the replication tree, rooted at the failed leaf peer, has to be updated. During this period (from the point of failure to the update of all indexing peers in the replication tree) a query directed towards the failed leaf peer will be evaluated as a false positive. This will increase search overhead to some extent.

To efficiently deal with frequent join and leave of a leaf peer, indexing peers should advertise their index information at

constant intervals. Any advertisement from a child peer should be delayed until the end of the interval. The interval length can be used to tradeoff index update delay with network overhead due to frequent advertisements.

III. Analysis

In this section we will provide an analytical bound on the levels of indexing hierarchy that will allow query routing in $O(\log N)$ hops, where N is the total number of peers in the system. For this analysis we will use B to denote branching factor, R for replication factor and n_l as the number of peers at level l. Leaf peers reside at level 0 and the height (or maximum level) of the indexing hierarchy is h. Assuming these definitions we can calculate the total number of peers at level l as:

$$n_l = n_0 \left(\frac{R}{B}\right)^l = n_0 \alpha^l \tag{1}$$

and the total number of peers in the system as:

$$N = \sum_{l=0}^{l=h} n_0 \alpha^l = n_0 \frac{\alpha^{h+1} - 1}{\alpha - 1} \tag{2}$$

Hence the total number leaf peers in the system,

$$n_0 = N \frac{\alpha - 1}{\alpha^{h+1} - 1} \tag{3}$$

Now, the number of groups at level l is R^l. So, the average number of peers in a group at level l, say m_l, is:

$$m_l = \frac{n_l}{R^l} \tag{4}$$

Combining equations (1), (2), (4) and using $\alpha = \frac{R}{B}$ we obtain:

$$m_l = \frac{N(\alpha - 1)}{B^l(\alpha^{h+1} - 1)} \tag{5}$$

In practice, in an overlay network, the number of peers in a group will be close to m_l. And for efficient query routing we expect the number of peers in a group at level h to be $f \times \log N$, where f is a positive real number, in range $(0.5 \leq f \leq 2)$. So, replacing $m_h = f \times \log N$ in equation (5) we obtain:

$$B^h(\alpha^{h+1} - 1) = \frac{N(\alpha - 1)}{f \log N} \tag{6}$$

For practical values of α and h we can approximate $(\alpha^{h+1} - 1)$ with $(\theta \times \alpha^h)$ for some constant θ. Replacing this value in equation (6) gives:

$$h \approx \frac{1}{\log R} \times \log \frac{N \times (\alpha - 1)}{\theta f \log N} = O\left(\log \frac{N}{\log N}\right) \tag{7}$$

Thus we claim that if we can build and maintain an index hierarchy of hight $O\left(\log \frac{N}{\log N}\right)$ then we will be able to solve the Distributed Pattern Matching problem in $O\left(\log N + \xi \log \frac{N}{\log N}\right)$ time, where, $\xi = \varepsilon\kappa$, κ is the number of results required by a query, and ε is the amount of false positives introduced by the lossy aggregation scheme. For a system without any aggregation the value of ε would be equal to one. Experimental results presented in section IV-B places a bound of 1.3 on ε. The results also demonstrate that this bound is not dependent on the number of peers in the network.

IV. Experimental Evaluation

To measure the performance of the proposed system and to verify the concepts presented in this paper, we have developed a prototype of the system and have simulated the prototype with various parameter settings. This preliminary version of the implementation simulates a static P2P network confirming to the DPMS architecture. Though, the current implementation is not capable of simulating node-joins and dynamic topology maintenance, it has the ability to simulate peer failures. This allows us to evaluate the impact of replication on routing performance and hit rate of a query, in the presence of peer failures. The current implementation can easily be plugged into the PeerSim [3] framework. We intend to use PeerSim simulator for performing experiments with dynamic overlays. Each data point presented in this section has been calculated as an average of the statistical values obtained from 10 independent simulation runs and 3000 random queries per simulation run.

In these experiments, patterns advertised by the leaf peers are uniformly distributed over the pattern space. But, this is not the case for applications, like partial keyword matching or service discovery. In these applications the patterns are bloom-filters, constructed from a finite set of elements, e.g., n-grams for partial keyword matching, or attribute-value pairs for service discovery systems. As a result, the patterns are more likely to be concentrated at various regions in the pattern space, instead of being distributed uniformly over the pattern space. This property would allow better level of aggregation of the advertised patterns than the pure random case presented in this section.

In section IV-A, we identify the system parameters and performance metrics, and investigate the impact of various system parameters on these performance metrics. Then we focus on the scaling behavior of the system with network size, in section IV-B. This allows us to place some bound on the value of ε (see section III) for specific parameter settings. Finally, in section IV-C, we present the experimental results to illustrate the impact of various levels of replication on the system's tolerance to peer failure.

A. Parameter tuning

DPMS uses aggregation for reducing index storage size at the peers contributing to the higher levels of the indexing hierarchy. The level of aggregation introduces a trade-off between query routing efficiency and index storage size at peers. A higher level of aggregation results into a lower level of storage overhead, a higher level of information loss in the aggregates, and a decrease in the query routing efficiency, and vice versa. In this section we intend to find a balance between these two conflicting interests.

Table I summaries the system parameters and their values or ranges used for parameter tuning.

The first four parameters in table I define a particular point in the sample space, containing all possible instances of DPMS hierarchies. These experiments are dedicated for analyzing the impact of different system parameters on query routing performance and storage overhead. Hence, we have chosen R=1. This allows us to separate the performance characteristics of the system from the fault-tolerance behavior.

The last three parameters, on the other hand, influences query routing efficiency. The aggregation process strives to achieve an aggregation ratio equal to A without introducing more than $(W - O)$ don't cares in any aggregate. The impact of parameter W (pattern width) on query routing performance and level of aggregation is intuitive though not trivial. Experimental results demonstrate that, for a fixed value of O, increase in W increases query routing accuracy while decreasing index storage requirement at peers.

Param.	Value(s)	Description
B	4	Branching factor
R	1	Replication factor
H	4	Maximum level
N	4092	no. of peers in the system
P	10	number of patterns advertised by a leaf peer.
A	0.6	target aggregation ratio[1]
O	10, 20…60	Minimum number of non-X bits in an aggregate
W	80, 100…200	Pattern or aggregate width

TABLE I

SYSTEM PARAMETERS AND THEIR VALUES USED FOR THE PARAMETER TUNING EXPERIMENTS

The performance metrics analyzed in this section are:

- **first-hit probes**: The number of peers that are being probed before the first match is found.
- **Avg. probes/hit**: The average number of probes required for each hit, in cases where multiple matches are present. We have traced up to 20 matches.
- **Indexing overhead (IO)**: This metric gives a measure of the extra storage space requirement introduced by the indexing hierarchy. Indexing overhead is measured as the ratio of the total number of aggregates in the system to the total number of patterns advertised by the leaf peers.

$$IO = \frac{\text{no. of aggregates (at the inexing peers)}}{\text{no. of patterns (at the leaf peers)}} \quad (8)$$

- **Effectiveness of aggregation (EA)**: This metric provides with a measure of the amount of reduction in the total number of aggregates achieved with the aggregation mechanism. The following equation is used to measure this quantity:

$$EA = 1 - \frac{\text{no. of aggregates with aggregation}}{\text{no. of aggregates without aggregation}} \quad (9)$$

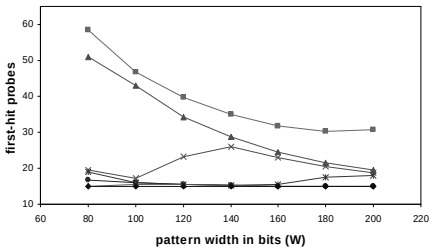
[1] A is computed as the ratio of the number of aggregates in the out-list to the number of patterns/aggregates in the in-lists, received from B children.

Fig. 5(a) and 5(b) show the impact of O and W on query routing accuracy. While Fig. 5(c) and 5(d) show the impact of O and W on storage overhead. By analyzing the curves in these figures, we can infer the followings:

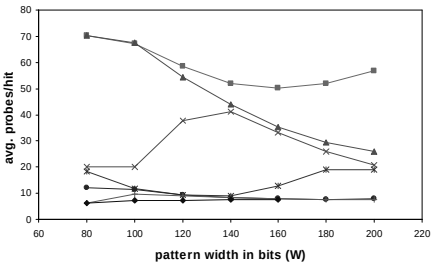
- Query routing accuracy increases with increase in minimum number of non-X bits (O) in the aggregates. This effect is intuitive. With higher number of original bits we have more information and lower probability of false positives.
- Indexing overhead increases with increase in the number of non-X bits (O). An increase in non-X bits means lower number of don't care bits are allowed in the aggregates, which implies less space for aggregation. This results in a higher number of aggregates in the system. This justifies the previous observation as well.
- For $O = W$ case there is no aggregation in the system and the number of false positives is zero, which gives the best possible values for query routing metrics. For this case, the indexing overhead is 4, while effectiveness of aggregation is 0.
- With $O = 60$ and $W = 80$, at most 20 don't care bits are allowed in the aggregates. With this restriction no aggregation takes place. This justifies the sharp rise in indexing overhead for $O = 60$ curve at point $W = 80$ (see Fig. 5(c)).
- Not all bits of a pattern are required for query routing with high accuracy. Query routing accuracy is almost the same for $O = 50$ and $O = W$ cases, whereas, indexing overhead for $O = 50$ case is only 65% of $O = W$ case.
- For $O = 10$ and $O = 20$ curves query routing accuracy increases with increase in W. This is because we are using both positional value and content of each bit while matching a query to an aggregate. When W increases, possible positions of non-X bits increases, which reduces the probability of false positives. Hence, the decrease in the number of hops.
- For a fixed value of O, indexing overhead decreases with increase in pattern-width (W). Given two random patterns, the probability that they will match on a given number of bits increases with W. This results into higher probability of aggregation and lower indexing overhead.

B. Scaling Behavior

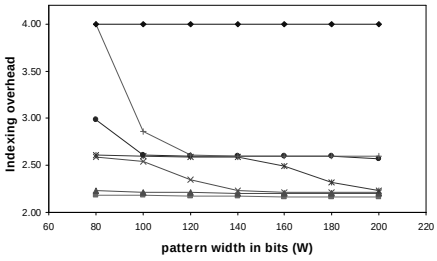
In this section we analyze the scaling behavior of DPMS with growth in network size. Based on the observations presented in section IV-A we have chosen $W = 180$ and $O = 50$ for the experiment presented in this section. The number of peers (N) in the system has been varied from around 8000 to 16000 while keeping the number of peers per group at the highest level of indexing hierarchy in the range of $\log N$ and $2 \log N$. We have used $R = 1$ and $B = 4$ to remain compatible with experiments present in the previous section. The value of H was set to 5 to accommodate all the peers in the indexing hierarchy, without violating the above mentioned constraints.



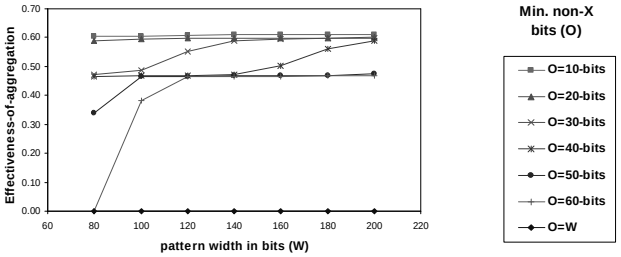
(a) First hit probes



(b) Avg. probes per hit



(c) Indexing overhead



(d) Effectiveness of aggregation

Fig. 5. **Impact of O and W on routing performance and storage overhead**

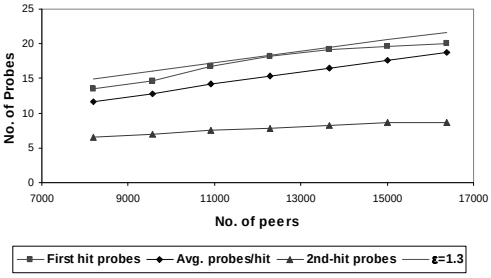


Fig. 6. **Scaling behavior**

Fig. 6 presents the impact of network size on query routing performance. This figure presents the curves for the first hit probe and the average probes per hit metrics according to definitions presented in section IV-A. The metric *2nd hit probes* represents the average number of additional probes required for the second hit. The first hit probe includes the cost of flooding ($O(\log N)$) the peers in a group at the hight level of the indexing hierarchy. This justifies the gap between the first hit probes and the second hit probes.

The upper bound curve is a plot of the following equation:

$$probes = no.\,of\,root\,peers + \varepsilon \log \frac{N}{\log N} \qquad (10)$$

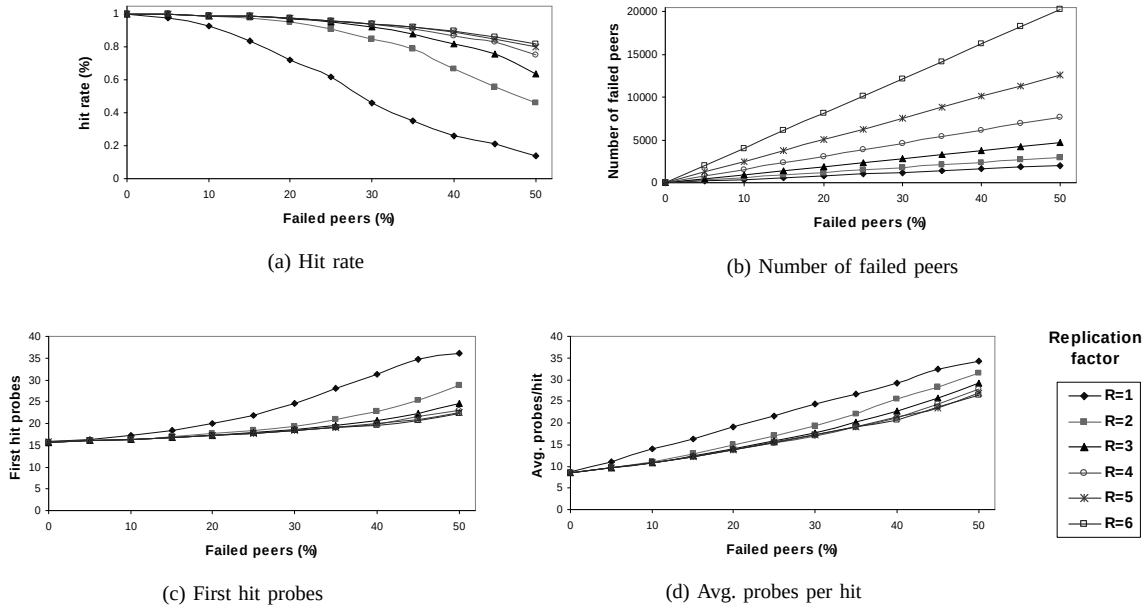
The value of ε was found to be 1.3. This implies that the value of ε is not dependent on the number of peers in the system.

It has also been revealed from the experiment that the indexing overhead (IO) and effectiveness of aggregation (EA) does not depend on the number of peers in the system. These metrics are dependent on the maximum number of levels (H), and the replication factor (R). An increase in H (and/or R) increases IO (and decreases EA), as the number of aggregates increases while the number of leaf patterns remains the same. For this experiment neither the value of H nor the value of R has been varied, which kept the ratio of the number of advertised patterns to the number of aggregates in the system the same. We obtained an average IO of 3.09 and EA of 0.48 for each of the sample points. The value of EA indicates that the aggregation process allows us to reduce the number of aggregates in the system by 48% from no aggregation case.

C. Fault tolerance

This section presents the impact of replication on routing performance in the presence of peer failure. For the experiments of this section we have varied the replication factor R from 1 to 6 while keeping all other parameter at a constant value. We have used $H = 4$, $B = 4$, $O = 50$ and $W = 180$. For these settings the number of peers in the system varied from about $4,000$ (for $R = 1$ case) to $40,500$ (for $R = 6$ case). For each value of R we have deactivated (i.e. removed) upto 50% of the peers from the system in a step of 5%, and have executed 3000 queries on the rest of the peers for each case (simulation run). Each point in the curves in Fig. 7 represents the average of 10 such simulation runs.

Like the previous experiments, we have used the first hit probes (Fig. 7(c)) and the average number of probes per hit (Fig. 7(d)) as the metrics for measuring query routing performance. For the previous experiments there were no failure of peers, and so all the matches to a query could be discovered. But, for the experiments in this section this is not true anymore. Due to the failure of peers we may have leaf peers that are unreachable from some peers at the highest



(a) Hit rate

(b) Number of failed peers

(c) First hit probes

(d) Avg. probes per hit

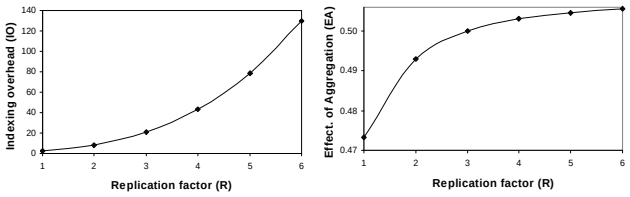
Fig. 7. **Impact of replication and peer failure on routing performance and hit rate of a query**

level of the indexing hierarchy. To measure this phenomena we have used the hit rate (Fig. 7(a)), i.e., the average percentage of matches that are discovered by a query. The impact of replication on the overall storage overhead in the system is presented in Fig. 8.

By analyzing the curves in Fig. 7 and Fig. 8 we can infer the followings:

- Without any replication ($R = 1$ case) the hit rate reduces drastically with increase in the percentage of failed peers (see Fig. 7(a)). Failure of an indexing peer, in this case, makes all of the leaf peers in its subtree unreachable. Routing efficiency is also low in this case, as many probes are wasted (i.e., evaluated as false positive at a parent of a failed indexing peer) in trying to route queries toward unreachable peers.
- Routing efficiency and hit rate increases with increase in R. This increase in hit-rate and routing efficiency (i.e., a decrease in the number of probes) diminishes with higher values of R (e.g. $R = 5$ or $R = 6$).
- The downside of replication is the exponential increase in the indexing overhead (see Fig. 8(a)). However effectiveness of aggregation increases with increases in R (see Fig. 8(b)). Based on the equations in section III and the definition of EA (see equation (9)it can be shown that the value EA tends to $1-A^{h-1}$ (actual aggregation ratio) as R tends to infinity. This justifies the gradual decrease in EA in Fig 8(b).

In the light of the experiments presented in this section we can conclude that, replication is necessary for improving reliability of the proposed system. A replication factor in the range of 2 or 3 can satisfy the need of most applications, assuming the peer failure rate is less than 30%.



(a) (b)

Fig. 8. **Impact of replication on storage overhead**

V. Related Works

Existing solutions for pattern matching [4], [10] in centralized environment, hold linear relationship with the number of advertised patterns (or *text* according to pattern-matching literature) to be matched. This implies flooding, for an equivalent solution to the DPM problem.

From architectural point of view, Secure Service Discovery Service (SSDS) [11] is the closest match to DPMS. Like DPMS, SSDS uses Bloom filter and aggregation. However, index distribution in SSDS is through a tree-like hierarchy of indexing nodes, in contrast to the lattice-like hierarchy used by DPMS. SSDS does not use any replication in the indexing hierarchy. Higher level nodes in SSDS index tree handle higher volume of query/advertisment traffic and the system is more sensitive to the failure of these nodes. Another major drawback of SSDS, compared to DMPS, lies in its aggregation mechanism. SSDS uses bitwise logical-OR for index aggregation. The aggregation scheme adopted in DPMS (explained in section II-C) retains unchanged bits from constituent patterns and provides more useful information during

query routing.

Unstructured systems ([2],[1]) identify objects by keywords. Advertisements and queries are in terms of the keywords associated with the shared objects. Structured systems, on the other hand, identify objects by keys, generated by applying one-way hash function on keywords associated with an object. Key-based query routing is much efficient than keyword-based unstructured query routing. But, the downside of key-based query routing is the lack of support for partial-matching semantics. Unstructured systems, utilizing blind search methods (like flooding[2] and random-walks [14]), can easily be modified to facilitate partial-matching of queries, and in general to solve DPM problem. But, due to the lack of proper routing information, the generated query routing traffic would be very high. Besides, there would be no guarantee on search completeness.

Many research activities are aimed at improving the routing performance of unstructured P2P systems. Different routing hints are used in different approaches. In [7] routing is biased by peer capacity; queries are routed to peers of higher capacity with higher probability. In [22] and [20] peers learn from the results of previous routing decisions and bias future query routing based on this knowledge. In [9] peers are organized based on common interest, and restricted flooding is performed in different interest groups. Many research papers ([7], [22], [12], etc.) propose storing index information from peers within a radius of 2 or 3 hops on the overlay network. All of these techniques reduce volume of query traffic to some extent, but do not provide guarantee on search completeness.

Bloom filter is used by many unstructured P2P systems for improving query routing performance. In [12], each peer stores Bloom filters from peers one or two hops away. Experimental results presented in [12] show that logical OR-based aggregation of Bloom filters is not suitable for aggregating information from peers more than one hop away. In [17] each peer store a list of Bloom filters per neighbor. The i^th Bloom filter in the list for neighbor, say M, summarizes the documents that are $i-1$ hops away via M. A query is forwarded to the neighbor with a matching Bloom filter at the smallest hop-distance. This approach aims to find the closest replica of a document with a high probability.

Schmidt et. al. [18] have presented an approach, named Squid, for supporting partial keyword matching in DHT-based structured P2P networks. They have adopted space-filling-curves to map similar keywords to numerically close keys. Squid supports partial prefix matching (e.g. queries like compu* or net*) and multi keyword queries. But, Squid does not have provision for supporting true inexact matching for queries like **net**.

In general DHT-techniques ([19], [16], [15] etc.) are not suitable for solving partial keyword matching problem (and DPM problem) for two reasons. Firstly, DHT-techniques require to partition the key-space into non-overlapping regions and to assign each region to a peer bearing an ID from that region. But from pattern matching perspective it is quite difficult to partition even one dimensional pattern (or key) space into non-overlapping clusters, while preserving the notion of closeness in patterns. Secondly, DHT-techniques cannot handle *common keywords problem* [13] well. Popular n-grams like "tion" or "ing" can incur heavy load on the peers responsible for these n-grams, resulting into unequal distribution of load among the participating peers.

VI. Conclusion and future works

In this paper, we have defined the Distributed Pattern Matching (DPM) problem and have presented a scalable solution, DPMS, to this problem. DPMS can be used to solve problems like wildcard searching (for file-sharing P2P systems), partial service description matching (for service discovery systems) etc. The number of probes for finding each match (or hit) in DPMS is in $(O(\log \frac{N}{logN}))$, where N is the number of peers in the system. For moderately stable networks DPMS provides guarantee on search completeness. DPMS can exploit the heterogeneity of peers and can handle churn problem by maintaining replicas for indexing peers. In DPMS, peers do not need to have any global view of the system and require to communicate with only a constant number of peers (B children, R parents and optionally some neighbors).

The main drawback of the proposed system is the storage overhead introduced by hierarchical indexing and replication. Experimental results presented in this paper demonstrates the worst possible values for the indexing overhead (i.e., for random case). For most application, there exists some bias (similarity) in the advertised patterns, which can enable much higher level of aggregation and hence much lower levels of indexing overhead. Another problem in DPMS stems from the leave/join of leaf peers. Leave/join of indexing peers has only a local effect. But, leave/join of a leaf peer can result into cascaded update along its replication tree. This problem can be mitigated by using periodic and differential updates of index information between the indexing peers. This update latency will not hamper the normal operation of the system other than degrading the query routing performance.

The experimental results presented in this paper demonstrate the strengths of DPMS. It has been shown that only a little fraction of the peers are probed for finding each match. The effectiveness of replication has also been presented. We are extending this research to demonstrate the applicability of DPMS in different application domains, and to show the impact of a bias in the advertised patterns on indexing overhead.

References

[1] Fasttrack peer-to-peer technology, http://www.fasttrack.nu/.

[2] http://www.gnutella.com, gnutella website.

[3] Peersim peer-to-peer simulator, http://peersim.sourceforge.net/.

[4] Amihood Amir, Ely Porat, and Moshe Lewenstein. Approximate subset matching with don't cares. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 305–306, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[5] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[6] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. In *Proceedings of the Allerton Conference*, 2002.

[7] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *Sigcomm*, 2003.

[8] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *SIGCOMM Conference*, August 2002.

[9] Edith Cohen, Amos Fiat, and Haim Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. In *IEEE INFOCOM*, 2003.

[10] R. Cole and R. Harihan. Tree pattern matching and subset matching in randomized $o(n \log^3 m)$ time. In *Proc. 29th ACM STOC*, pages 66–75, 1997.

[11] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz. An Architecture for a Secure Service Discovery Service. In *Mobile Computing and Networking*, pages 24–35, 1999.

[12] Mei Li, Wang-Chien Lee, and Anand Sivasubramaniam. Neighborhood signatures for searching p2p networks. In *IDEAS*, pages 149–159, 2003.

[13] Lintao Liu, Kyung Dong Ryu, and Kang-Won Lee. Supporting efficient keyword-based file search in peer-to-peer file sharing systems. In *GLOBECOM*, 2004.

[14] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS*, 2002.

[15] Petar Maymounkov and David Mazi. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer-Verlag, 2002.

[16] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, 2001.

[17] S. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *INFOCOM*, 2002.

[18] C. Schmidt and M. Parashar. Flexible information discovery in decentralized distributed systems. In *Proceedings of HPDC*, WA, USA, June 2003.

[19] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, pages 149–160, San Diego, California 2001.

[20] D. Tsoumakos and N. Roussopoulos. Adaptive probabilistic search for peer-to-peer networks. In *3rd IEEE Intl Conference on P2P Computing*, 2003.

[21] Spyros Voulgaris, Márk Jelasity, and Maarten van Steen. A robust and scalable peer-to-peer gossiping protocol. In *Proceedings of the Second International Workshop on Agents and Peer-to-Peer Computing (AP2PC)*, 2003.

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

[23] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.