Efficient and Scalable Query Routing for Unstructured Peer-to-Peer Networks

Abhishek Kumar Jun (Jim) Xu Ellen W. Zegura
College of Computing,
Georgia Institute of Technology,
{akumar,jx,ewz}@cc.gatech.edu

***Abstract*— Searching for content in peer-to-peer networks is an interesting and challenging problem. Queries in Gnutella-like unstructured systems that use flooding or random walk to search must visit $O(n)$ nodes in a network of size n, thus consuming significant amounts of bandwidth. In this paper, we propose a query routing protocol that allows low bandwidth consumption during query forwarding using a low cost mechanism to create and maintain information about nearby objects. To achieve this, our protocol maintains a lightweight probabilistic routing table at each node that suggests the location of each object in the network. Following the corresponding routing table entries, a query can reach the destination in a small number of hops with high probability. However, maintaining routing tables in a large and highly dynamic network requires non-traditional mechanisms.**

We design a novel data structure called an Exponentially Decaying Bloom Filter (EDBF) that encodes such probabilistic routing tables in a highly compressed manner, and allows for efficient aggregation and propagation. The search primitives provided by our system can be used to search for single keys or multiple keywords with equal ease. Analytical modeling of our design predicts significant improvements in search efficiency, verified through extensive simulations in which we observed an order of magnitude reduction in query path length over previous proposals.

I. Introduction

The mechanisms for searches in peer-to-peer (p2p) networks are severely constrained due to the distributed nature of content indices and highly dynamic membership of hosts in the network. In the context of peer-to-peer networks, searching is equivalent to routing a query to a node in the network that hosts content matching the query. Routing in traditional networks has endeavored to achieve deterministic and complete routing in the face of (relatively infrequent) network changes. On the other hand, content-based routing for peer-to-peer networks has to grapple with the large size of routable content and frequent changes in the network membership. This rules out the conventional solution of building and maintaining a per-destination (object) routing table through an exchange of routing updates. However, relaxing the requirements of completeness and determinism in routing enables some interesting solutions that we explore in this paper.

The existing solutions to achieve content-based routing can be divided into two broad families. The first family of *structured p2p networks* consists of solutions that impose a particular structure on the overlay network (e.g., [1], [2], [3], [4], [5]). The regularities in this structure are then exploited to efficiently maintain and query a global data-structure such a Distributed Hash Table (DHT).

The second family of *unstructured p2p networks* comprises Gnutella-like networks that do not impose any structure on the overlay network [6]. The default search mechanism in Gnutella is to blindly forward queries to all neighbors within a certain number of hops. Although this mechanism handles network dynamics very well, search through blind flooding is quite inefficient. This has motivated a host of studies proposing various enhancements to search in unstructured networks. Major improvements include replacing the blind flooding with a random-walk [7] or an expanding ring search, tailoring the network construction to achieve properties of small world graphs [8], reflecting the capacities of heterogeneous nodes in topology-construction [9], and caching pointers to content located one hop away [9]. All of these proposals (except for caching) retain the "blind" nature of query forwarding in Gnutella. In other words, the forwarding of queries is independent of the query string and does not exploit the information contained in the query itself. The keywords in the query are used only for searching the local content index.

The objective of this work is to design an efficient query-routing mechanism for unstructured peer-to-peer networks. We propose to build probabilistic routing tables at nodes, constructed and maintained through an exchange of updates among immediate neighbors in the overlay. These routing tables use a novel data structure — the Exponential Decay Bloom Filter (EDBF) — to efficiently store and propagate probabilistic information about content hosted in the neighborhood of a node. The amount of information in an EDBF (and the number of bits used to store this information) decreases exponentially with distance. Such exponential decrease in information with distance restricts the impact of network dynamics to the neighborhood of any departing or newly arriving node. The *Scalable Query Routing (SQR)* mechanism we design uses hints obtained from these probabilistic routing tables to forward queries. The use of probabilistic hints provides a significant advantage over the completely blind nature of existing mechanisms, translating into large reductions in the average number of hops over which a query is forwarded before it is answered.

We evaluate our design through both analytical modeling and simulations. Our analysis provides good insight into the complex process of probabilistic information dissemination in

0-7803-8968-9/05/$20.00 (c)2005 IEEE

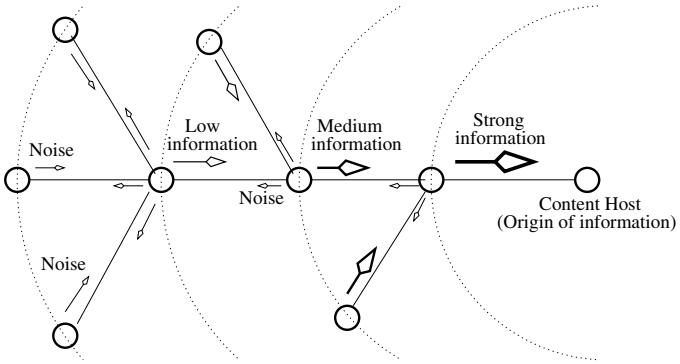


Fig. 1. Exponentially decaying information in the neighborhood of a node. Arrow sizes represent the amount of information or awareness about content hosted at the node on the right extreme of the figure. Notice how the noise, depicted as small arrows, is present everywhere but is dominated by the information as we get closer to the host.

random graphs and its use for efficient routing. Using the analytical model, we derive expressions for query success probability and the expected query path length. Predictions from this model closely match our observations of the system in simulation.

Our simulations cover a number of proposed systems and demonstrate the performance advantages of using a probabilistic routing mechanism like SQR. SQR achieves one to three orders of magnitude reduction in query path-length over the existing and proposed systems. These improvements are observed to be consistent over a range of content replication models.

The rest of this paper is organized as follows. An overview of SQR is presented in the next section, followed by a detailed description of its design in section III. Our model of SQR and its analysis is presented in section IV. In section V, the predictions from our analysis are verified and the performance of SQR is studied through extensive simulations.We also study the benefits of coupling SQR with various topology adaptation mechanisms. A brief discussion of our solution being an instance of a more general paradigm for information processing in dynamic networks is presented in section VI. We review related work in section VII and conclude in section VIII with pointers to future work.

II. Overview

In this section we present a brief overview of our solution, deferring a detailed description to the next section. The key idea of our Scalable Query Routing (SQR) scheme is captured in Fig. 1. SQR propagates awareness about content hosted at a node in the neighborhood of the node, with the amount of information decreasing exponentially with the distance from the hosting node. The node on the extreme right of the figure is hosting a piece of content and originating information about it. As a consequence, nodes close to the origin have strong information about the content at the origin. The strength of this information decreases with distance until it becomes indistinguishable from noise towards the extreme left of the figure. As we will show, this exponential decay not only reduces the amount of communication overhead to a minimum, but also has a blind decay feature that allows the aggressive compression of the route update, making the scheme highly efficient. Surprisingly, even such a small amount of route information can significantly improve the search operation.

This concept of propagating highly compressed information regarding contents is realized through the design of a novel data structure called Exponential Decay Bloom Filter (EDBF). Intuitively, the EDBF is a data structure that allows us to vary the number of bits used to store probabilistic information. As the distance from the node hosting the object (source of information) increases, fewer bits are used to represent information about the direction in which the object is located. At sufficiently large distances the number of bits used becomes much smaller than 1 and is dominated by random noise due to collisions in hashing, thus effectively restricting the size of the "aware" neighborhood[1]. Section III-A describes the design of EDBF in detail and discusses its succinct encoding of routing information.

Routing in general, has two major components, the first of which is concerned with the construction and maintenance of routing and forwarding tables. In SQR the routing table is a set of EDBF's, one corresponding to each link. Nodes send route advertisements during the setup of (overlay) links. Incremental updates are used to maintain the consistency of advertised information. The second component of routing is to use these routing/forwarding tables to forward queries. While forwarding a query, the EDBF corresponding to each of the neighbors is examined. The query is forwarded to the neighbor advertising the maximum amount of information about the queried object. Within the aware neighborhood, this action propels the query towards the node hosting the queried object. Since EDBF is a "noisy" data structure, at nodes outside the aware neighborhood, the maximum information is just a random accumulation of noise and following this random noise corresponds to a random walk. Thus from a global view, each query follows a random walk until it hits a node within the aware neighborhood, and then quickly gravitates towards the node that is originating information about the target object. The cost of each query decreases as the size of the aware neighborhood increases.

Through our design of SQR in this paper, we make a case for discarding the maintenance of precise routing information, in favor of approximate, but considerably efficient, probabilistic routing information maintained through EDBFs. Our evaluation through analysis and simulation establishes the scalability of SQR and paves the way for wide-area deployment of query routing with propagation of routing updates. The next section gives a complete description of the various components

[1]Aside from its use in SQR, EDBF can have a number of independent applications. Just as a Bloom Filters can efficiently represent a set, an EDBF can efficiently represent a fuzzy set (i.e., a set with probabilistic membership). Such a data structure can be used in situations where probabilistic information needs to be efficiently represented, e.g. stochastic inference algorithms in network measurement, distributed web-cache coherence, etc.

of our solution and their synthesis to provide scalable query routing.

III. Design

In this section we describe the details of our query routing mechanism. We begin with the design of the Exponential Decay Bloom Filter (EDBF), a data structure for space-efficient representation of probabilistic information. We then describe the routing protocol that builds and maintains routing tables using EDBFs and end this section with a description of the use of these probabilistic routing tables for forwarding queries. An expanded version of the design, including a detailed description of implementing keyword searches and discussions of several additional issues, is presented in our technical report [10].

A. Design of the Exponential Decay Bloom Filter

The EDBF is an extension of the traditional Bloom filter (BF) [11] to encode the probabilistic forwarding table. In particular, it encodes the approximate hop-count distance from the node to any given object in the network. In the following, we introduce EDBF after a brief review of the design of traditional Bloom filter.

A Bloom filter is a data structure for approximately answering set membership questions. It employs a fixed number (k) of hash functions and an array of bits A initialized to all 0 at the beginning. When inserting an element x in a BF, all k hash functions $h_1(.), \cdots, h_k(.)$ are evaluated simultaneously over x and the bits in A indexed by $h_i(x)$, $i = 1, 2, ..., k$, are set to 1. A query for the element y in a BF looks at the bits in A indexed by $h_1(y), \cdots, h_k(y)$ and returns a *yes* if and only if all these bits are 1.

An EDBF uses a fixed number (k) of hash functions h_1, h_2, ..., h_k too. As in a normal Bloom filter, all k hash functions are used for insertion. However, given a query for an object x, EDBF simply returns $\theta(x) = |\{i|A[h_i(x)] = 1, i = 1, 2, ..., k\}|$, the number of 1's in the filter. How to interpret this number $\theta(x)$ is application-specific. When EDBF is used to encode the probabilistic forwarding table in SQR, $\theta(x)/k$ roughly indicates the probability of finding x along a specific link in the overlay.

The ED part of the name EDBF comes from the fact that, when there is no noise, the number of bits $\theta(x)$ decays exponentially (at a constant rate $1/d$) with its distance (in terms of hops) from the node where the object x is stored. In other words, when there is no noise, at a node one hop away from the object x, $\theta(x)$ is approximately k bits, at a node two hops away, $\theta(x)$ is approximately k/d, and so on. While advertising routing information to downstream neighbors, nodes reset each of the bits in the EDBFs received from upstream neighbors with a probability $(1-d)$, thus achieving an exponential decay by a factor of d. The forwarding scheme (described later) implements the semantics of following the strongest signal, thus implicitly retracing the trail of the exponentially decaying bits advertised by the node hosting the object. Note that noise does exist due to collisions in hashing, i.e., some of the k bits indexed by $h_1(x)$, $h_2(x)$, ..., $h_k(x)$ can be set to 1 by other objects hashed to the same index. Therefore, the noise will impact our accuracy of predicting this distance, which will be studied in section IV.

Note that exponential decay is not the only conceivable type of decay. Other alternatives exist, e.g., *linear decay* in which the $\theta(x)$ value reduces linearly with distance from the node that hosts x. We however choose exponential decay for SQR because only it has an essential property of "blindness". Specifically, the process by which bits are flipped from 1 to 0 is totally independent of the association of that bit with any particular object x. In other words, the decision on whether to decay a bit or not is independent of the information as to which objects cause a bit to be set at what distances, and therefore such information need not be remembered by EDBF. This blindness property leads to extraordinary succinctness of EDBF and reduces the communication costs for route updates by orders of magnitude as compared to previous attempts at using Bloom Filters for content based routing [12], [13], [14], as discussed in section V. It will be clear that the Exponential Decay part is a critical innovation to overcome the scalability barrier of Bloom Filters without compromising the goal of achieving routing efficiency.

B. Creation and Maintenance of Routing Tables

SQR employs bandwidth-efficient (and therefore scalable) mechanisms for encoding and maintaining the probabilistic forwarding tables using EDBFs (see figure 2). At a node with degree l, its forwarding table consists of l EDBF data structures of the same array size, each of which corresponds to one of its neighbors and contains highly-compressed summary information regarding contents that can be reached through that neighbor. As shown in the next section, information contained in these EDBF tables will allow the query for an object x to be forwarded to the correct neighbor (on the shortest path to x) with high probability. In addition to these EDBF's, each node also keeps a list of contents that it hosts locally.

We have mentioned before that to make the scheme scalable, information regarding content x will decay exponentially with the hop-count distance to the node that hosts x. This is achieved by each node only propagating $1/d$ of its information (its EDBF's and EDBF encoding of its local contents) to its neighbors, through a blind decay process. As discussed before, the blind decay allows the encoding and propagation of route updates to be very bandwidth-efficient. This bandwidth-efficiency of route updates is further improved by using two known techniques with adaptations: (1) compressed EDBF mentioned above (2) delta compression, i.e., a full route advertisement at the beginning (when a new link is set up) and periodic "delta updates" thereafter.

1) Route Update Creation: Each node sends a route advertisement (in the form of an EDBF) to its neighbors at the time of connection setup, and maintains its consistency by periodic updates. The initial advertisement is created by taking the union of all advertisements received from neighbors other

```
Create Local EDBF (given local content X):
      // Populate local EDBF A.
 1. ∀x ∈ X
 2.       Set bits A[h_1(x)], ..., A[h_k(x)] to 1;

Create Update (for neighbor j):
    // Copy all the bits from the local EDBF A into
    // the update U_j.
 1. U_j ← A;
    // Decay the information received from all neighbors
    // other than j by a factor of d, and add the
    // surviving bits to U_j.
 2. ∀i ∈ neighbor_list, i ≠ j
 3.       ∀r ∈ {1, ··· , m}
 4.            if(A_i[r] == 1)
 5.                 with probability 1/d, U_j[r] ← 1;
 6. Return U_j;
```

Fig. 2. Algorithms for creating updates in SQR.

than the target neighbor, allowing this combined advertisement to decay by the decay factor d, and finally taking a union of the result with the local EDBF. Subsequent updates are created by periodically creating a fresh route advertisement for each neighbor and using delta compression to communicate the difference of this fresh advertisement with the actual information previously communicated to the neighbor.

The baseline algorithm for constructing the route update from a node to one of its neighbors b through link l is shown in Fig 2. Procedure `Create Local EDBF` takes the list X of keywords (or content labels) corresponding to locally hosted content and inserts it into an EDBF[2] A. In procedure `Create Update`, information received in the form of EDBFs from neighbors other than b is first aggregated by merging all such EDBFs into a single EDBF through a bitwise-OR operation. The bits in this "Union EDBF" are randomly reset to zero so that only $1/d$ of of the bits survive this decay operation. In other words, the union of updates received from all neighbors other than j is computed and "attenuated" by a factor of d before propagation. The local EDBF is propagated without attenuation.

Note that in our merging operation above, information (EDBF) received from a neigbor j will not be advertised back to j. This is analogous to "split horizon with poisoned reverse" in classical distance vector routing [15]. In fact, poisoned reverse in SQR is even more important because while in the distance vector algorithm the poisoned reverse makes a difference only when there is a node/link failure, in a P2P network the routable entities are shared objects that exhibit significantly higher dynamics. However, split horizon does not protect against all forms of loops. Fortunately, the exponentially decaying nature of information in SQR implies that the count to infinity problem manifests itself as a "decay to infinitely small amount of information", thus significantly restricting the impact of routing loops.

[2]The local EDBF is created when the node first joins the network and modified suitably whenever the set of locally hosted objects changes.

2) Update Compression and Transmission: Delta compression is the standard approach to further reduce the information that needs to be communicated between neighboring nodes for route updates. In SQR, nodes keep track of the cumulative information conveyed to their neighbors. Updates can be composed by taking just the difference between the EDBF table up to the last update and the freshly composed EDBF using the procedure `Create Update`.

Recent work on Compressed Bloom Filter by Mitzenmacher [16] has shown that if we reduce the number of hash functions k slightly but make the array much larger (resulting in a much sparser array) such that after compression the size remains n, the false positives in such a compressed Bloom filter can be much better than the optimal in a non-compressed version. This result also applies to EDBF as the sparseness of the array, caused by a sharp decay factor, produces an ideal candidate for compression. For example, if the decay factor d is 8 then only about 1/8 of the bits in the array are 1. Using *Arithmetic Coding* [17], an uncompressed EDBF of size m, with a fraction λ of total bits set to 1, can be compressed to a size $L = mH(\lambda)$, where H is the entropy function given by $H(\lambda) = -\lambda \log_2 \lambda - (1-\lambda)\log_2(1-\lambda)$. This can be seen as the communication cost of the first update received by a new node joining the network.

The procedure to send updates first inspects if the neighbor is a new one, in which case it sends the complete update. For existing neighbors, an incremental update[3] is sent by computing the XOR of the current update U_j with the previously sent information. All updates are compressed using arithmetic coding [17] before being sent reliably using TCP. The Delta updates are likely to be much more sparse than the initial updates, and hence, will benefit even more from arithmetic coding before final transmission of the update. At the other end, the procedure to receive updates first inspects if an EDBF A_i corresponding to the neighbor i exists. If not, this must be a new neighbor and the update is simply stored as A_i. Otherwise, the update is treated as an incremental one and A_i is modified suitably by computing its bitwise XOR with the new update.

The above design achieves our primary objective of efficiently maintaining probabilistic information about the content stored in the neighborhood. Due to the random resetting of bits during forwarding of advertisements, the impact of a node's advertisements on another node's routing table reduces exponentially with the distance between the two. An initial update is sent whenever a node acquires a new neighbor. Incremental updates for each neighbor j are created periodically but sent asynchronously only when the delta update contains at least δ bits that are 1, where δ is a parameter to be tuned.

```
Forward Query (given query Y):
      // Forward previously seen queries to neighbor i, .
      // chosen randomly from neighbor_list.
1. if( Seen Query(Y))
2.    Deliver Query(Y,i);
3. else
   //Forward previously unseen queries to the neighbor
   // with the maximum information about this query.
4.      Θ ← Lookup (Y);
5.      Pick i such that θ_i = max(Θ);
6.      Deliver Query(Y,i);

Lookup (given query Y):
1. ∀i ∈ neighbor_list
2.      ∀q ∈ {1, ··· , k}
3.          θ_i+ = A_i[h_q(y)];
4. Return Θ;        /*Θ = {θ_i}*/
```

Fig. 3. Algorithms for forwarding queries in SQR.

C. Query Forwarding

The algorithm for forwarding queries in SQR is summarized in figure 3. If the query is satisfied locally, it is answered. Otherwise, if the TTL of the query has not expired, the current node has to pick a neighbor to whom the query is forwarded. Our solution can be seen as a simple greedy algorithm with enhancements for loop avoidance and tie-breaking. If the query was previously seen, it is forwarded to a randomly chosen neighbor. Otherwise, the query is processed as follows. The query string x is looked up in the EDBF associated with each neighbor of the node. For each neighbor i, the result of this lookup is an "indicator" value θ_i, which is the total number of bits set to 1 in locations indexed by $h_j(x), j \in 1...k$. The query is forwarded to the neighbor with the highest indicator value after decrementing the TTL by 1. Ties are broken randomly. In an ideal situation, with the aware neighborhood extending throughout the network, and an exponentially increasing number of bits pointing towards the origin of the information about any object, queries will follow the shortest path to their destination. However, due to limited aware neighborhood size (to save on the cost of creating and maintaining the routing tables), and false positives in the EDBF, some additional issues need to be addressed before our description of the forwarding algorithm is complete.

The first issue we need to consider is that of "noise" in the indicator values as computed from the EDBF data-structure. Due to collisions in hashing, some of the bits that contribute to a total of θ_i in advertisement A_i, could be caused by other unrelated keywords hashing to the same locations as $h_q(y)$. When the query is far away from the origin of information about an object, the information would have decayed to much less than one bit, and almost all bits corresponding to the query are likely to be such false positives. The neighbor with the largest θ-value is likely to be a local maximum for such random noise. Thus forwarding to the neighbor with the highest indicator value is equivalent to forwarding to a randomly chosen neighbor while outside the aware neighborhood.

Even inside the aware neighborhood, where the number of bits due to the source of the information is non-negligible, the noise due to false positives might sometimes overwhelm the correct information, thus leading to a wrong forwarding decision. We develop an analytical model in section IV that captures this behavior and derive expression for the probability of making such mistake at various distances from the node hosting the target object.

We assume that queries contain unique identifiers (as in Gnutella [6]) that are cached[4] at intermediate nodes when seen for the first time. Thus, queries revisiting a node can be easily identified. As discussed above, the behavior of SQR outside the aware neighborhood is equivalent to random walk. It is possible that nodes are visited multiple times in such a random walk. To correctly implement the behavior of random walk, such "loop-back" queries should be forwarded to a randomly chosen neighbor.

Another way to understand the intuition behind this behavior is to look at it from the perspective of an intermediate node that sees a loop-back query. The first time this query was seen at this node, it must have been forwarded to the neighbor with the maximum number of bits corresponding to the query. Now, since the query has somehow propagated back to this node, the previous routing decision was wrong. In such a case, the amount of information in the routing tables is too low to make any inference about the location of the queried object, and a randomly chosen neighbor is as good a candidate as any to receive this query[5].

IV. Analysis

In this section we develop an analytical model of SQR. The main purpose of this model is to answer through analysis two questions: (1) Where does the performance gain in routing latency come from? and (2) At what communication cost? These will be answered in sections IV-B and IV-C respectively, after a discussion in section IV-A of the underlying topology that we assume. The proofs of all theorems in this section are omitted due to lack of space.

A. P2P network topology for our study

Both metrics under study, the routing latency and the communication cost, are functions of the underlying P2P network topology. In this section, we describe a typical topology (close to a random regular graph), that will be assumed in the rest of the paper. We emphasize, however, that our analytical methods and insights from the analysis are orthogonal to this topology and can be easily adapted to other topologies.

[3]Even in the absence of content dynamics, the randomness of the decay process will cause different bits to be reset each time an update is constructed. To avoid this problem, we use *pseudo-random decay* that ensures consistency in the decay process.

[4]The cost of storing id's of recently seen queries is negligible even at nodes with a high query throughput.

[5]In our experiments, we tried more sophisticated alternative mechanisms like choosing the neighbor with maximum bits among the untried neighbors, but did not observe any performance improvement over the much simpler protocol of randomly forwarding all loop-back queries.

We model the underlying network topology as a graph of n vertices, with average degree c at each node. We make a further simplifying assumption that for small i, the number of nodes exactly i hops away from an arbitrary node is $n_i = c(c-1)^{i-1}$. These assumptions are not precisely accurate unless the graph is a tree rooted at the arbitrarily chosen node, but they are a good approximation in the region where i is chosen so that $\sum_{j=1}^{i} n_j << n$. Note that we could model the network as a regular graph, which will make the above assumption exact. However, we feel that the random topology mentioned above is close to the actual Gnutella topology in which peers randomly attach themselves to a set of other nodes. Therefore, our analysis is only an approximation when assuming this random topology. For ease of exposition, we make a second simplifying assumption in setting the size of content index (i.e., the list of content labels) at each node be a constant. This assumption does not hold in real peer-to-peer networks, but the distribution of content index sizes is orthogonal to our analysis and abstracting it out simplifies the modeling exercise.

B. EDBF – Impact on routing latency

Since SQR uses an enhanced random walk, the latency of query routing is proportional to the number of hops traversed by the query from its source to a node that hosts matching content. Our model of routing latency is thus clearly dependent on the accuracy of the forwarding tables stored in the form of EDBFs. The accuracy of EDBFs in turn is reflected in the false positive rate of the EDBF under system parameters. Once we model the probability of false positives, we use it as a building block to construct a Markovian model of the overall process of query routing.

1) False positives in an EDBF: Let the number of hash functions used in the EDBF be k and and the decay factor be d, as before. Let S be the total number of bits at all nodes in the network caused by one piece of content at node I. In other words, S is the total cost of all updates that originate at I and propagate throughout the network. With the above assumptions, S is given by:

$$S = n_1 k + n_2 \frac{k}{d} + n_3 \frac{k}{d^2} + n_4 \frac{k}{d^3} + \cdots \leq \sum_{j=1}^{i} n_j \frac{k}{d^{j-1}} + \frac{k}{d^i} \left[n - \sum_{j=1}^{i} n_j \right] \quad (1)$$

Due to the symmetry of the model, the total number of bits at all other nodes caused by a node I, is equal to the total number of bits caused by all other nodes at node I. Let m be total size of the array storing the EDBF. Since I has c neighbors, each of the c EDBF's representing probabilistic routing tables have S/c uniformly random bits set to 1. By Whang statistics [18], we get the total number of 1's in each of the EDBF's to be:

$$m_1 = m \left(1 - e^{-S/cm}\right) \quad (2)$$

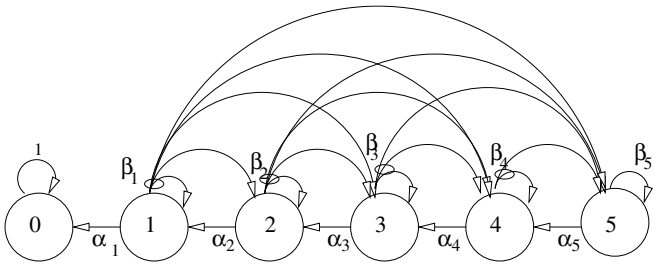


Fig. 4. A Markov chain model of routing using SQR.

The fraction of bits set to 1 is given by $\lambda = m_1/m = \left(1 - e^{-S/cm}\right)$. We will use this value of λ to compute the probabilities of false positives.

States in the Markov chain. The probabilistic nature of routing using SQR can be modeled as a Markov chain. The Markov chain in our model, shown in figure 4 has six states, numbered 0 through 5. For any satisfiable query (i.e., one for which the corresponding object is hosted by some node in the network) we classify all n nodes in the network as being in exactly one of these states. A node is in state i if its shortest distance from the node hosting the object is i hops, with all nodes farther than 5 hops away also counted as being in state 5. Let the number of nodes in state i be n_i, with $n_5 = n - \sum_{i=0}^{4} n_i$. Our choice of 6 states represents the approximation that at a distance of 5 hops from a node hosting an object, the amount of information about this object in EDBF based routing tables is negligible. In other words nodes 5 or more hops away from the node hosting the object have approximately the same (negligible) amount of information about the location of this object. This assumption is valid as long as $k/d^4 << 1$, i.e.. the average number of bits that survive 4 successive decays by a factor of d, out of the k bits set at the origin node, is much smaller than 1.

Transition probabilities. Forwarding operations in SQR correspond to state transitions in the Markov chain in figure 4. Recall that the forwarding rule in SQR is to pick the neighbor with the strongest signal for a new query, and to forward previously seen queries to a randomly chosen neighbor. At a node X, the information from the upstream neighbor X_1 is stronger than the noise[6] from all other neighbors $X_2, X_3, \cdots$, then the query is forwarded to X_1 and the system makes a transition from state i to $i-1$. The probability for this event is α_i. Theorem 1 derives an expression for calculating the value of α_i. If the noise from a non-upstream neighbor of X dominates the signal from the upstream neighbor (with probability $\beta_i = 1 - \alpha_i$), the query gets forwarded to a neighbor at an equal or higher distance from the node hosting the object. Due to the uniform random nature of noise in EDBF, the source of this misleading noise is equally likely to be in any of the states i and above. Hence, we distribute the total transition probability β_i out of state i into β_{ij} going

[6]If the information is exactly the same as the noise from $t \geq 1$ neighbors but stronger than the noise from all other neighbors, the query is forwarded to X_1 with probability $1/t = 1$. This case is also covered by theorem 1.

to state j ($j \geq i$) in proportion to the node population of each of these states, such that $\sum \beta_{ij} = \beta_i$.

Justifications for the use of Markovian modeling. For our design to qualify as a Markov process, we need to show that the forwarding process is memoryless in the sense that the forwarding operation performed on a query is independent of the path it has traversed. This is mostly true except in the case when a query visits the same node a second time. Since queries are never returned to the node they are received from, the probability of such a loop is quite small. Recall that such queries are forwarded to a randomly chosen neighbor. The randomness of this choice minimizes the impact of this memory, allowing us to ignore this situation and make the assumption of memorylessness in our model.

Computing the transition probabilities. The following theorem formulates a way to calculate the values of different transition probabilities in our model.

Theorem 1: The transition probability from state i to $i-1$, $i \geq 1$, denoted by α_i is given by:

$$\alpha_i = \sum_{r=a}^{b} \frac{1}{a-b+1} \left(Pr[X > Y]^{r-1} + \sum_{t=1}^{r-1} \frac{1}{t+1} Pr[X = Y]^t Pr[X > Y]^{r-t-1} \right) \quad (3)$$

Where $[a, b]$ is the range of possible node degrees, and X and Y are binomial random variables, picked from the binomial distributions $B(k, 1/d^{i-1} + \lambda - \lambda/d^{i-1})$ and $B(k, \lambda)$ respectively.

Corollary 1: The transition probability from state i to states $j \geq i$ and above is given by $\beta_{ij} = \frac{n_j}{n_{tail}} \beta_i$, where $\beta_i = 1 - \alpha_i$ and $n_{tail} = n_i + n_{i+1} + \cdots + n_5$.

C. Computing the cost of forwarding using SQR

Using the model developed above, we can compute the number of steps required to route a query to its destination. Let $B = (b_{i,j})$ be a 6×6 matrix representing the transition probabilities among various states. In this matrix, $b_{i,i-1} = \alpha_i, \forall i \geq 1$ and $b_{ij} = \beta_{ij}, \forall i \geq 1, j \geq i$. Since nodes are associated with different states corresponding to their shortest distance from the origin, there can be no transition from state i to state $i-2$ and below. Thus $b_{ij} = 0$, $\forall j \leq i-2$. Finally queries reaching state 0 are answered correctly and stay in this state hereafter[7]. Thus $b_{0j} = b_{j0} = 0$, $\forall j \neq 0$ and $b_{00} = 1$. The entries of matrix B are summarized in figure 5.

Let $T_{i,0}$, $i = 1, 2, ..., 5$, denote the number of steps it takes for a query that starts at a node in state i, to **first** encounter the target (i.e., state 0). Let T denote the number of steps for a query that starts at a node chosen uniformly at random, to first encounter the destination. Since the probability of a query originating in state i is n_i/n, it is easy to verify that

$$E[T] = \sum_{i=1}^{5} \frac{n_i}{n} E[T_{i,0}] \quad (4)$$

[7]Our model considers a query successfully answered from the point where a query reaches the first node that has matching content.

1	0	0	0	0	0
α_1	β_{11}	β_{12}	β_{13}	β_{14}	β_{15}
0	α_2	β_{22}	β_{23}	β_{24}	β_{25}
0	0	α_3	β_{33}	β_{34}	β_{35}
0	0	0	α_4	β_{44}	β_{45}
0	0	0	0	α_5	β_{55}

Fig. 5. The matrix B.

Parameter	Symbol	Value
Filter Width	m	$2^{18} = 256kbits$
No. of hash functions	k	64
Decay Factor	d	8

TABLE II
PARAMETERS OF EDBF USED IN SQR

Now it remains to compute $E[T_{i,0}]$. Let $C = (c_{i,j})$ be a 6x6 matrix such that all its entries are identical to those of B, except that $c_{0,0}$ = 0 (note that $b_{0,0}$ = 1). The values of $E[T_{i,0}]$ are characterized by the following theorem:

Theorem 2: Let the matrix $D = (d_{i,j})$ be defined as $D = C(I - C)^{-1}(I - C)^{-1}$. Then $E[T_{i,0}] = d_{i,0}$.

V. EVALUATIONS

In this section we present a simulation-based evaluation of SQR and compare its performance with other mechanisms for query routing in unstructured peer to peer systems. The various systems covered by our simulations are summarized in table I. We broadly classify the systems as either having a flat topology or a hierarchical topology. Within these broad classes, different mechanisms for query forwarding can be used, giving rise to a large number of combinations. The set of systems we study is by no means complete, but our choices are representative of the body of prior work in this area and cover both existing systems as well as recent ones proposed in the research literature.

The parameters for the EDBF used in SQR are listed in table II. The simulations use a 2,500 node network, unless noted otherwise. The primary performance results can be summarized as follows. For the same routing overheads, the query response rate improves by up to two orders of magnitude for flat topologies (section V-A), matching very well with the previous analysis. Simulations over hierarchical topologies (in section V-B demonstrate the compatibility of SQR with various topology construction and maintenance mechanisms. The drastic reductions in the overheads of routing table construction and maintenance achieved through SQR's use of EDBF are hilighted in section V-C). Finally, a study of the impact of content replication in section V-D suggests that the benefits of SQR carry over to various models of content replication.

A. Query response rate

The original Guntella protocol (version 0.4) constructs a topology with relatively similar node degrees, with clients actively seeking neighbors till they have at least min_neigh (=3

System	Topology	Search mechanism	# neighbors per node [min,max]
Flood	flat (Gnutella V0.4)	Scoped flooding	[3,8]
Random	flat (Gnutella V0.4)	Random Walk	[3,8]
OHR	flat (Gnutella V0.4)	Random Walk with one hop replication of pointers	[3,8]
SQR	flat (Gnutella V0.4)	This paper	[3,8]
Ultrapeer	explicit hierarchy (Gnutella V0.6)	Scoped flooding among Ultrapeers	Leaf → Ultrapeer [1,3] Ultrapeer → Ultrapeer [3,8] Ultrapeer → Leaf [1,100]
GIA	implicit hierarchy (GIA)	Random walk with biased forwarding	[3,128]
SQR+GIA	implicit hierarchy (GIA)	SQR with heterogeneous nodes	[3,128]

TABLE I

SYSTEMS USED IN OUR EVALUATION.

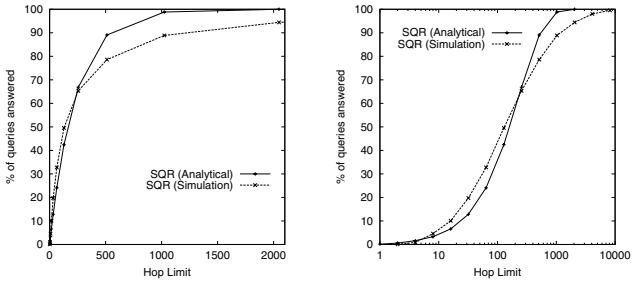
in our simulations) neighbors and accept neighbors till they have at most max_neigh (=8 in our simulations) neighbors. We use this flat topology to evaluate a number of systems. The first system, *Flood*, uses flooding to propagate queries. We also simulate the system *Random* that replaces flooding with random walk. Random walks have been proposed as a more efficient replacement for scoped flooding [7], with the assumption that the Gnutella V0.4 topology resembles a random graph. Replication of pointers to content hosted one hop away [9], [13] is another enhancement to Gnutella V0.4. The system OHR in our simulations uses random walk with *One Hop Replication* of pointers.

To obtain a common metric that captures the resource usage of both flooding and random walk based approaches, we use a generalization of TTL called Hop Limit (HL). For random walk based systems, that do not create multiple copies of a message, HL is the same as TTL. However, if scoped flooding is used, and a query with hop limit h is received at a node with f neighbors, it is forwarded to the remaining $f-1$ neighbors with a hop limit of $(h-1)/(f-1)$ appropriately rounded up or down[8] so that the hop limit of all the flooded messages adds up to $h-1$. The use of Hop Limit allows us to capture the cost of both random-walk and flooding based mechanisms on a common scale.

The set of object labels and queries are picked at random from a uniform distribution. We only simulate queries for content actually hosted in the network, with the understanding that queries for non-available content will fail after using up the maximum allowable hops, irrespective of the search mechanism in use. While all query routing mechanisms achieve a 100% success rate with sufficiently high hop limits, only the superior query routing mechanisms can achieve successful delivery of queries with low hop limits. Varying the hop limit allows us to study the profile of resource usage by various systems required to achieve different query success rates.

Figure 6 shows the plot of number of queries answered vs. the initial hop limit. Figure 6(a) and 6(b) are for flat topologies and figure 6(c) shows the results for hierarchical topologies (discussed further in section V-B). Log scale is used on the x-axis in figures 6(b) and 6(c) to better capture the difference in the amount of resources consumed by different mechanisms. Among the systems with flat topology, flooding is the most inefficient, followed by plain random walks. One hop replication of pointers improves the efficiency by an order of magnitude and using SQR improves the efficiency by an additional order of magnitude.

[8]More precisely, the hop limit of each of the messages forwarded to neighbor i, where $i \in \{1, \cdots, f-1\}$, is set to $\lfloor h-1/f-1 \rfloor + 1$ if $i \leq (h-1) \ mod \ (f-1)$ and to $\lfloor h-1/f-1 \rfloor + 1$ otherwise.



(a) Normal scale. (b) Log scale on x-axis.

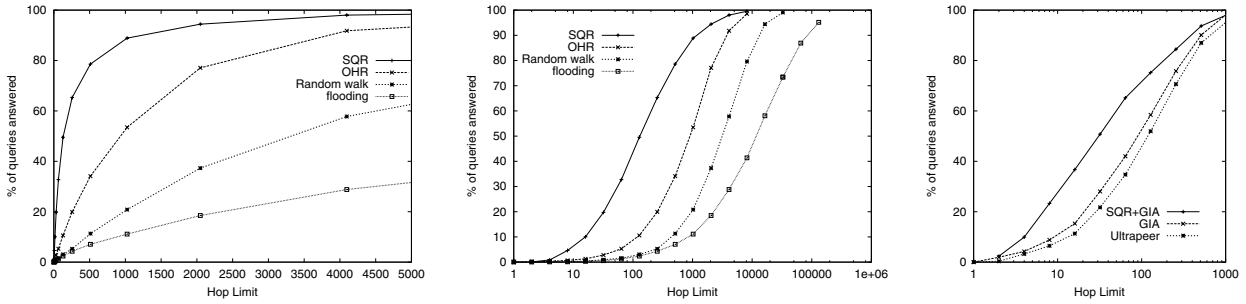
Fig. 7. Comparison of analytical predictions with simulations.

Metric	Analysis	Simulation
Mean	232	221
Median	161	130

TABLE III

COMPARISON OF ANALYTICAL PREDICTIONS & SIMULATION RESULTS.

Comparing analytical predictions with simulation results. Figure 7 compares the predicted behavior of SQR from our model in the previous section with the observed performance of system in simulation. The analytical curve corresponds to numerical computations based on theorems 1 and 2. The simulation curve is the same as in figure 6(b), described in section V-A. Recall that the analysis in section IV models the performance of SQR in random graphs. This is an approximation of the topology constructed by Gnutella



(a) Flat topologies.

(b) Flat topologies, log scale on x-axis.

(c) Hierarchical topologies, log scale on x-axis.

Fig. 6. Number of queries answered vs. hop limit, with one query per node.

Capacity	10000x	1000x	100x	10x	1x
Fraction	0.1%	4.9%	30%	45%	20%

TABLE IV

CAPACITY DISTRIBUTION OF CLIENTS IN HIERARCHICAL TOPOLOGIES.

Mechanism	SQR	OHR	Rhea et al. [14]
Overhead (Bytes)	187	262	30,000 to 90,000

TABLE V

ROUTING OVERHEADS PER OBJECT

V0.4, used in our simulations. In spite of this approximation, and other simplifying assumptions made during the modeling, the two curves have a close fit. Similarly, in table III, the values of the mean and median of the query length distribution obtained from our simulations are quite close (within 20%) to the analytical predictions.

B. Query response rate for hierarchical topologies

The second class of systems we simulate includes GIA [9] and Gnutella with Ultrapeers [19], [20], that have the explicitly stated design goal of exploiting the heterogeneity in node capacities, in terms of processing capabilities and/or network bandwidth. They try to build a topology that reflects the capacity of a node in the degree of connectivity of the node, with higher capacity nodes corresponding to high degree vertices in the overlay graph. The topology of Gnutella V0.6 can be seen as a two layer hierarchy with a strongly connected layer of Ultrapeers (high capacity) that provide a flooding based search service to a large number of directly attached leaf nodes (low capacity). The topology adaptation algorithm of GIA [9] also achieves a similar situation of a small number of well connected high capacity nodes to which a large number of low capacity nodes attach themselves. Table IV lists the capacity distribution of nodes in hierarchical topologies. This distribution is derived from a measurement study of node capacities [21], and has been used in previous work on hierarchical topologies [9]. For Gnutella V0.6, the nodes with capacities $\geq$1000x are chosen as Ultrapeers.

Figure 6(c) shows the query response rate of various systems that use advanced topology adaptation mechanisms to construct hierarchical topologies. SQR+GIA is SQR with GIA's topology adaptation mechanism. The simulations in [9], verified independently by our own simulations, demonstrate the superiority of GIA's topology construction[9] over that of Gnutella V0.6. The superior performance of SQR+GIA supports our claim that the efficiencies of SQR carry over to any network irrespective of the topology.

C. Routing overheads

Table V summarizes the overheads of various mechanisms. The routing overhead of SQR is the cost of building and maintaining the EDBF based routing tables. An upper bound on S — the number of bits propagated across all links in the network by one object — was derived in equation 1. Assume that the fraction of bits that are one in the EDBF's, defined as λ, is 0.1. If the average degree of nodes (c)is 5, the total number of hash functions (k) is 32 and the decay factor (d) is 8, the value of S from equation 1 is 320 bits. Since λ, the fraction of bits in the EDBF's, is 0.1, for every bit that is 1, nine others have to be zero, bringing the cost up by a factor of 10 to 3200 bits. However, since updates are compressed using arithmetic coding, the actual cost is about $3200H(\lambda)$ bits, where H is the entropy function. For $\lambda = 0.1$, $H(\lambda) = 0.469$, and this cost evaluates to 1500 bits or 187 bytes per object. We emphasize that this is the total cost of routing updates caused by one object over the entire network and not the cost of a single update.

The cost of one hop replication of pointers to content hosted at each node is higher. An empirical analysis of 1.3 million file names traced from query responses on a Gnutella network shows that the average file name is 52.4 bytes long[10].

[9]The gap between the curves for GIA and Ultrapeer increases for larger sizes of the simulated network. However, routing tables in SQR make the simulations very memory intensive and do not scale to large sizes easily. For the sake of uniformity, we simulate a network of 2,500 nodes for all systems.

[10]This is closely linked to the fact that searches in Gnutella are keyword searches over content filenames

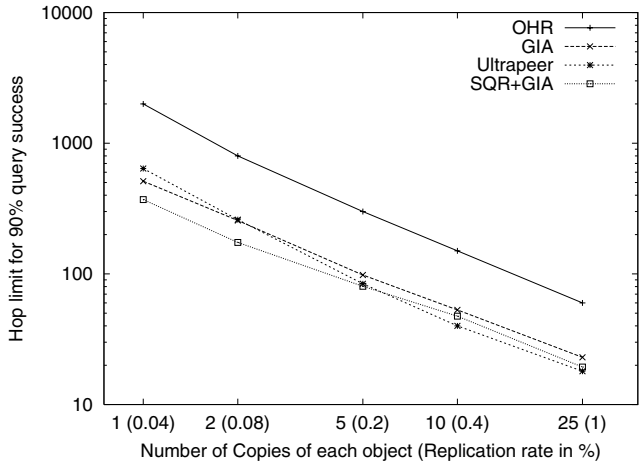


Fig. 8. Hop limit for 90% query success vs. replication rate. Notice the log-scale on both axes.

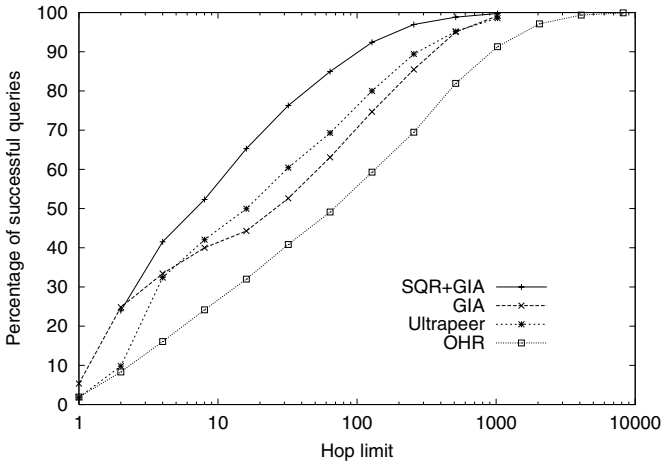


Fig. 9. Success rate of queries vs. hop limit in a network with replication according to the Zipf distribution. Notice the contrast with figure 6(b).

Replicating these filenames at each neighbor in a network with average degree of 5 will cost $5 \times 52.4 = 262$ bytes.

Rhea et al. [14] propose the use of Bloom Filters to enhance the search performance, assuming that the basic search functionality is provided by a deterministic location and routing mechanism, such as an idealized directory service or a DHT-based search. The proposed protocol creates an attenuated Bloom filter – which is just a collection of d Bloom filters, with the ith Bloom filter at a node being associated with content published by nodes i hops away from the node. While forwarding a query, the ith filter associated with each link is examined. If a match is found in an attenuated Bloom filter, the query is forwarded on the corresponding link. Otherwise, the $(i+1)$th Bloom filters at each link are examined. On finding no match in any of the d Bloom filters, the default search algorithm is used. Routing updates are flooded to nodes up to d hops away from each source of content. The costs of propagating this information is 30 to 90 kilobytes per object [14]. We emphasize that although this was an acceptable cost for the target application in [14], such high cost renders infeasible the option of using or extending this work to provide an underlying global query routing service[11].

D. Impact of Replication

Having multiple instances of the same object at different locations in the network improves the efficiency of all query routing mechanisms by cutting short the average query path. In this section, we demonstrate that SQR benefits equally from the effects of replication. The first model of content replication we use assumes a uniform (constant) rate of replication for all content in the system. While this is quite unlikely to occur in a real system, choosing a uniform replication rate allows us to isolate the effect of specific rates of replication over different systems. For simplicity, the content popularity model in these simulations is the same as the content replication model.

Figure 8 demonstrates the effect of replication on different query routing systems. The x-axis represents the number of copies of each object (the uniform replication rate) in a 2500 node network. The y-axis represents the resource usage to attain a 90% query success rate in such a network. Curves for all systems have an almost constant negative slope in this log-log plot. The relative advantages for different systems are roughly maintained across different replication rates[12] , supporting our claim that SQR benefits equally from the effects of replication.

The second model we use for replication of content is based on the Zipf distribution, frequently used to model the replication of objects on the web and in peer-to-peer systems [22]. In a set of objects replicated according to the Zipf distribution, the ith most replicated object has $1/i$ times as many replicas as the most replicated object. In our experiments, we use a replication rate of 10% for the most replicated object. The success rate of queries is plotted against the initial hop limit in figure 9. Notice how the lower half of the curves differ from those in figure 6. The faster initial growth in success rate can be attributed to the large number of replicas of the most replicated item. However, the top end of the curves look the same as in figure 6. This portion corresponds to the queries that take a lot of steps to get answered, most of them being queries that target rare or unreplicated objects with low rank

[11]The update mechanism designed by Rhea et al. [14] has to remember several bytes of information for *each individual bit* in the Bloom filter, such as the ID of the source node and its distance from the current node. Object-deletions in their mechanism are quite complex and potentially even more expensive than insertions. EDBF was specifically designed with properties of "blindness" and exponential decay, which in turn enables a simple update protocol that can handle both insertions and deletions of objects with equal ease and yet is cheaper by more than two orders of magnitude.

[12]In figure 8, the system Ultrapeer shows a faster performance improvement with increasing replication rate. This is an artifact of the relatively small number of Ultrapeers in the simulated topology (5% of 2500 nodes), where the explicitly hierarchical topology of Gnutella V0.6 starts looking like a client-server system at high replication rates. These advantages are absent at network sizes above 10,000 nodes and we observed Ultrapeer to perform worse than GIA in simulated networks of such sizes.

in the Zipf distribution.

VI. Discussion

In this section, we briefly discuss a new paradigm of **Distributed Coordinated Data Streaming**. Our design of SQR is an application of this paradigm to the problem of query routing. Data streaming[13] has been touted as a viable solution for measuring and monitoring of high-speed links in large networks [24]. We believe that the scope of networking problems to which Data-Streaming can be applied, can be expanded to include highly dynamic environments such as peer-to-peer and sensor networks. A common scenario in such networks is that each node produces a large number of events, not all of which can be efficiently advertised to all nodes of the network[14]. Frequent changes in the membership of individual peers or content hosted by them in peer-to-peer networks, and consecutive observations at a sensor node in a sensor network are examples of such large event streams. The information associated with this event stream can be seen as a data-stream. In coordinated data streaming, the stream of data from each node is filtered and compressed by its neighbors, the neighbors' neighbors, and so on. Each node performs such streaming locally, over the data provided by its neighbors, and maintains a synopsis data structure to store the streaming results. Querying this synopsis data-structure provides local solutions which can then be composed to perform complex operations in the overall network. Local forwarding decisions using the EDBF tables in SQR can be seen as instances of such local queries, which when composed, implicitly through the handoff of the query to the neighbor with the strongest information, perform the complex operation of query routing in a distributed fashion. Coordinated data streaming has applications in large and dynamic environments, and we plan to work on some of them in our future research.

VII. Related work

Structured p2p networks (e.g., [1], [2], [3], [4], [5]) have been proposed to solve the problem of efficient searching by imposing a particular structure on the overlay network and exploiting this structure to efficiently maintain and query a global data-structure such a Distributed Hash Table (DHT) storing a unique key associated with each data item through hashing. Proposals like pSearch [25] associate each piece of content and query with a vector in a Cartesian space embedded in the structured network.

The use of Bloom filters to store and propagate information about distributed content in overlay networks has been proposed earlier in various contexts [12], [13], [14], [26]. The work in the Gnutella development community [12], [13] has focused on specifying and implementing a protocol for constructing and maintaining a Bloom Filter based routing table for keyword queries. This body of existing work, although pioneering the idea of using Bloom Filters to represent routing information, suffers from a limited understanding of the effects of propagating such routing information. Indeed, the authors of [13], while specifying the Query Routing protocol for Gnutella, conclude by highlighting the need for a careful study before propagation of query routing tables is switched on. We argue that the Gnutella Query Routing protocol in its current form is not amenable to propagation of routing information, as the overheads will exceed the cost of flooding based search in even moderately dynamic networks.

GIA is a comprehensive framework for increasing the scalability of Gnutella-like systems by exploiting the heterogeneity in node capacities to build an implicit hierarchy in the system [9]. GIA has components for *dynamic topology adaptation, active flow control, biased random walk* for search and *one hop replication of pointers* to content. High capacity nodes have a higher degree and have pointers to a lot of content due to one hop replication of pointers and are visited preferentially due to the biased random walk. The danger of congestion at such high capacity nodes is mitigated by the active flow control mechanism. Our efforts, have been focussed on disseminating information about content as efficiently as possible, and then exploiting this partial information to route queries efficiently. The mechanisms for topology adaptation and flow control in GIA are orthogonal to our design and can be easily assimilated into an SQR based system. The routing tables in SQR have significantly larger amount of information than one hop replication of queries in GIA, and are cheaper to construct and maintain. Our evaluation shows that replacing the replication of pointers and biased random walk in GIA with SQR produces a system that retains all the benefits of GIA while improving the efficiency of query routing.

The use of routing indices in peer-to-peer networks was proposed by Crespo et al. [27]. Their scheme assumes that the content is classified under "topics" and nodes index the number of documents under each topic reachable through each of their immediate neighbors. History based systems typically cache some information from previous queries to "route" future queries more intelligently. For example, the adaptive probabilistic search mechanism proposed by Tsoumakos et al. [28] uses an adaptive "learning" algorithm to associate success probabilities for various queries along each of the immediate neighbors of the node. The use of probabilistic query forwarding, instead of deterministic flooding to all neighbors, is another way of reducing duplicate queries in the network and has been proposed by Kalogeraki et al. [29].

VIII. Conclusions

Routing in networks is a complex problem. But for the success of hierarchical routing, it would be impossible for the Internet to reach its present scale. Unfortunately the prerequisites for hierarchical routing such as prefix aggregation and a hierarchy in the network topology, are absent from the domain of query routing in peer-to-peer networks. In this work, we

[13] As a note of clarification, the term *data streaming* here has no connection with the transmission of multimedia data known as media (audio and video) streaming [23].

[14] Flooding all the data to all the nodes is clearly not scalable and bandwidth-efficient.

have explored the approach of spreading probabilistic information about the location of hosted content in its neighborhood, and then using this information for forwarding queries. The simple design of SQR makes it possible to analytically model the mechanisms for propagation of information and its use in forwarding, thus enabling better understanding of the design ideas and also allowing us to predict the performance of SQR. Simulation based comparison with various query routing mechanisms, under a wide variety of scenarios, establish the performance advantages of SQR and demonstrate the ease with which SQR can inter work with other system components such as topology construction. We expect a similar ease of integration with other system components such as flow control. While providing a set of mechanisms for query routing, SQR imposes little restriction on the possible policies that can be implemented. A case in point is the ease with which the semantics of keyword searches can be composed using SQR. The framework provided by SQR presents interesting possibilities of implementing high level semantics of trust, reliability, etc. using routing and forwarding policies. These are interesting issues that merit further exploration.

REFERENCES

[1] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *Proc. of ACM SIGCOMM '01*, 2001.

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proc. of ACM SIGCOMM*, 2001.

[3] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[4] B. Y. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," U.C. Berkeley Tech. Report UCB/CSD-01-1141, Tech. Rep., 2001.

[5] A. Kumar, S. Merugu, J. Xu, and X. Yu, "Ulysses: A Robust, Low-Diameter, Low-Latency Peer-to-Peer Network," in *Proc. of IEEE ICNP*, 2003.

[6] "http://gnutella.wego.com."

[7] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," in *Proceedings of IEEE Infocom*, 2004.

[8] S. Merugu, S. Srinivasan, and E. W. Zegura, "Adding structure to unstructured peer-to-peer networks: the role of overlay topology," in *Proceedings of Networked Group Communication (NGC).*, 2003.

[9] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *Proc. of ACM SIGCOMM'03*. ACM Press, 2003, pp. 407–418.

[10] A. Kumar, J. Xu, and E. W. Zegura, "Efficient and scalable query routing for unstructured peer-to-peer networks," Georgia Institute of Technology, Tech. Rep., 2004, available at http://www.cc.gatech.edu/ akumar.

[11] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *CACM*, vol. 13, no. 7, pp. 422–426, 1970.

[12] M. T. Prinkey, "An efficient scheme for query processing on peer-to-peer networks," aeolus Research, Inc., http://aeolusres.homestead.com/files/index.html.

[13] C. Rohrs, "Query routing for the gnutella network," lime Wire LLC, http://www.limewire.com/developer/query_routing/keyword routing.htm.

[14] S. C. Rhea and J. Kubiatowicz, "Probabilistic location and routing," in *Proc. of IEEE Infocom'02*, Mar. 2002.

[15] C. Huitema, *Routing in the Internet*. Prentice Hall PTR, 1999.

[16] M. Mitzenmacher, "Compressed bloom filters," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 604–612, 2002.

[17] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[18] K. Whang, B. Vander-Zanden, and H. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Transactions on Database Systems*, 1990.

[19] T. Clingberg and R. Manfredi, "Gnutella0.6," June 2002.

[20] A. Singla and C. Rohrs, "Ultrapeers: Another Step Towards Gnutella Scalability," 2002, version 1.0.

[21] S. Saroiu, K. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in *Multimedia Conferencing and Networking*, Jan 2002.

[22] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker., "Web caching and zipf-like distributions: Evidence and implications," in *Proceedings of IEEE Infocom*, Mar. 1999.

[23] D. S. Phatak and T. Goff, "A novel mechanism for data streaming across multiple IP links for improving throughput and reliability in mobile environments," in *Proc. IEEE INFOCOM*, June 2002.

[24] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Transactions on Database Systems (TODS)*, vol. 28, pp. 51–55, 2003.

[25] C. Tang, Z. Xu, and M. Mahalingam, "pSearch: Information Retrieval in Structured Overlays," in *ACM HotNets-I*, Oct. 2002.

[26] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay networks," in *Proceedings of the ACM SIGCOMM '02 Conference*, Aug. 2002.

[27] A. Crespo and H. Garcia-Molina, "Routing indices for peer-to-peer systems," in *Proc. of the 22nd International Conference on Distributed Computing Systems(IEEE ICDCS02)*, 2002.

[28] D. Tsoumakos and N. Roussopoulos, "Adaptive probabilistic search in peer-to-peer networks," in *Proc. of 3rd IEEE Intl Conference on P2P Computing*, 2003.

[29] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A local search mechanism for peer-to-peer networks," in *Proc. of the 11th ACM Conference on Information and Knowledge Management*, 2002.