# Approximate Index Routing: A Case for Content-based Peer-to-Peer Routing

Fengyun Cao   Jaswinder Pal Singh
Computer Science Department, Princeton University

*Abstract* – Object location in a distributed peer-to-peer system has been a challenging problem. Our goal is to achieve efficient and effective query routing while imposing little restriction upon the system structure. We propose the approach of content-based peer-to-peer routing, in comparison to the highly structured address-based routing schemes such as the distributed hash table (DHT) approaches. By decoupling object content and their locations, content-based routing retains the simplicity and flexibility of an unstructured peer-to-peer system.

We present one implementation of content-based peer-to-peer routing, namely Approximate Index Routing (AIR). Each AIR node maintains an independent summarization of the global index. The space consumption at each node is saved, while high routing accuracy is achieved by cooperation between nodes with different summarizations. Because of its simple structure, AIR is able to exploit application inherent optimization opportunities, such as resource heterogeneity and content locality. We show that AIR is able to scale to peer-to-peer systems with hundreds of millions of objects.

## 1. Introduction

Peer-to-peer (P2P) system has attracted dramatic attention in research communities. The nature of P2P systems can be characterized as a distributed system in which all nodes have equal capabilities and responsibilities and all communication is potentially symmetric. The applications built upon such P2P system can take advantage of resources (storage, computation cycles, content, human presence etc.) available at the edge of the Internet. P2P file-sharing systems, such as Gnutella [12] and KaZaa [13] are now one of the most popular Internet applications and have become a major source of Internet traffic [28] .

A fundamental challenge in distributed P2P systems is how to efficiently locate a relevant object stored in the system.

Popular P2P file sharing systems, such as Gnutella and KaZaa, adopt an *unstructured* approach in the sense that the search strategy imposes no specific restriction on system structure. The peers organize into an *ad hoc* overlay network and maintain no information about object locations. Query messages are flooded to all peers, generating high network traffic load. Gnutella introduces Time-To-Live (TTL) parameter that controls the scope of flooding. The result is that an object stored outside the query's horizon cannot be located. Unstructured P2P systems have the advantages of simplicity, flexibility and strong robustness. It also supports text-based search and partial queries. The major disadvantage is that the flooding-based search method is inherently unscalable.

In response to the scaling problem, a number of *highly structured* P2P systems have been proposed, among which are Chord [32] , Pastry [9] , Tapestry [33] , and CAN [20] . These systems maintain a global index of object locations as a distributed hash table (DHT). Objects in the system are associated with a unique ID (produced, for instance, by hashing the object content), and each node in the system is responsible for storing objects that map into a certain range of IDs. Nodes organize into a well-defined overlay network, according to the relationship between their storage responsibilities. A query is issued in the form of *lookup(objectID)*. Object ID is used to route the query message efficiently through the overlay network towards the node responsible for that ID.

Although DHT systems are elegant and highly scalable, the tight structure it imposes on the P2P system can restrict its applicability and deployment.

First, the random mapping between objects and their locations in DHT inherently conflicts with the geographic and content locality of a P2P application. Objects can be stored far from their frequent users, and at nodes that have neither contributed nor been interested in them.

Second, DHT scheme makes an implicit assumption of homogeneous node capability and responsibility. In contrast, it has been found ([25] ) that there is significant heterogeneity across peers that participate in real P2P file sharing systems. An incapable or highly volatile node can be problematic in maintaining the DHT routing invariant.

Third, in DHT systems, an object is associated with precisely one node. It is hard to locate multiple replicas of an object, or to search for multiple objects matching the same query. It is also impossible to know whether an object exists in the system before the query message is transmitted all the way to the node that is supposed to have the object *if it were in the system*. If such non-hit query is common, their transmission can result in high routing inefficiency.

Finally, DHT systems offer very scalable solution for "exact-match" queries on object identifiers. It is yet to be demonstrated that these systems can scalably support more powerful search semantics such as text-based search and partial queries.

In this paper, we explore the possibility to support efficient and effective query resolution in an unstructured P2P system. We call such scheme *content-based peer-to-peer routing*.

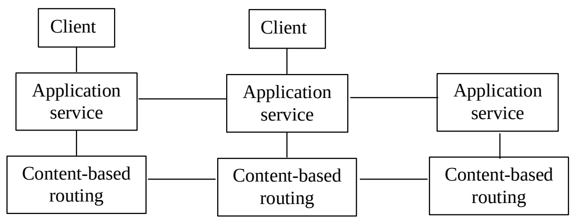


Figure 1: *Architecture of content-based P2P routing.*

The design philosophy for content-based P2P routing is to achieve efficient P2P routing while making little assumption or restriction on the application system. As illustrated in Figure 1, the routing algorithm is designed to be an independent module that is completely decoupled from other system components, such as storage management, and transparent to the upper layer functionalities. For generality and flexibility purposes, the methodology is to focus on routing framework design and leave policy-based decision-making as plug-in modules. Thus, the routing scheme has the ability to preserve or exploit application-specific properties, and is compatible with different application environments through appropriate configurations.

The basic search interface that content-based P2P routing supports is *search(index-value)*, in which index-value can be any indexable, descriptive feature of an object. The location of all objects that match the index-value will be returned. In comparison, the DHT scheme can be classified as *address-based routing* in that routing is based on specialized address information, e.g. object ID, associated with the message. By decoupling object content with their location, content-based P2P routing does not rely on any specific overlay network topology or object placement strategy. It differs from existing unstructured P2P system such as Gnutella in that query messages are selectively routed toward nodes that have the matching objects, rather than blindly flooded over the network.

In this paper, we present one implementation of content-based P2P routing, called Approximate Index Routing (AIR). The main idea in AIR is to achieve routing efficiency and system scalability by carefully balancing the usage of various system resources, such as storage space and network bandwidth. AIR is also able to exploit application-inherent optimization opportunities such as resource heterogeneity and content locality. We show that AIR can scale to P2P systems of hundreds of millions of objects, and can support a variety of applications. AIR is by no means the only way to implement content-based P2P routing, and we do not claim that it is necessarily better than the other P2P object location approaches. Rather, we expect AIR to provide preferable solutions to applications with certain characteristics, and that the techniques explored in AIR to be useful in other systems too. Our discussion is meant to be more illustrative than conclusive.

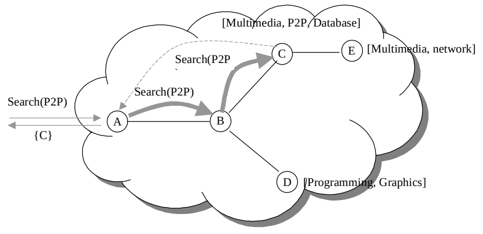


Figure 2: *Example of content based routing.*

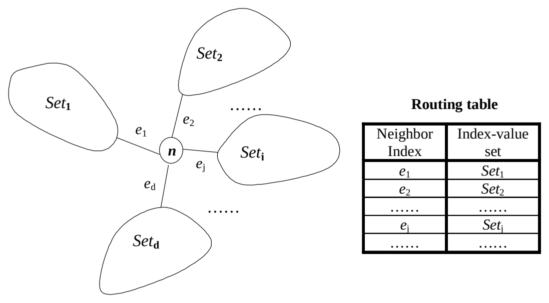
The rest of the paper is organized as follows: In section 2 an overview of content-based P2P routing is given and we briefly describe the basic system design of AIR. In section 3 AIR features and optimization techniques are analyzed and evaluated in detail. We discuss the scalability and applicability of AIR in practical world in section 4. Related work is reviewed in Section 5 and we conclude in Section 6.

## 2. System Design for Approximate Index Routing

In a content-based P2P routing system, nodes (peers) self-organize into an overlay network, as shown in Figure 2. A query message carrying index-value is forwarded hop-by-hop in this overlay network towards the location with the matching objects, possibly through shortest paths. The return message with identified locations is either propagated back through the overlay or directly sent to the initiating node.

### 2.1 Index routing

Index routing is one implementation of content-based routing. The overlay network in an index-routing system is an acyclic graph, i.e. a tree topology. Such tree can be a span built upon a mesh topology for resilience purpose. For simplicity, here we only discuss the tree topology. Each node in the system maintains a routing table as shown in Figure 3.



**Routing table**

| Neighbor Index | Index-value set |
|---|---|
| $e_1$ | $Set_1$ |
| $e_2$ | $Set_2$ |
| …… | …… |
| $e_j$ | $Set_j$ |
| …… | …… |

Figure 3: *Routing table in index routing system.*

A routing table has *d* entries, where *d* is the number of neighbors a node has. Each entry contains the set of index-values of objects stored in that neighbor's direction.

When receiving a query, the node checks the index-value membership against each entry in routing table, except the one for the direction that the message came from. Then, the message is forwarded to all directions that are indicated to lead to objects with the queried index-value.

In principle, as stated above, the index routing tables can be built by indexing on any feature (index-value) of objects that users want to query with. An index-value does not imply any information about the object's location. Neither is it required to completely or uniquely identify an object, e.g. an mp3 can have more than one keyword and different songs may share the same keywords. Therefore, the indexing routing scheme supports partial queries and allows explicit object replications.

The design of data structure of index sets in the routing tables and the algorithm of membership test have significant impact on performance of index routing. Specifically, to correctly route all query messages, the size of a routing table can be $O(N)$, where $N$ is the total number of index-values in the system. Thus, the potentially large routing table size and the complexity of membership test can be a major challenge for scalability of index routing.

## 2.2 Approximate Index Routing

In our work, we ask the question of whether P2P routing has to be performed with perfect accuracy. We observe that a little "false positive" routing, i.e. forwarding messages toward nodes that actually do not have the object does not hurt routing correctness: all the objects that match the query can still be located, although at the cost of extra network traffic load. On the other hand, "false negative" routing can fail to locate a queried object that exists in the system and is not tolerable. Based on this observation, we designed Approximate Index Routing (AIR) scheme, in which routing information is maintained at an approximate level, but using much less space. The space efficiency is achieved at the cost of a little false positive routing. Next, we describe AIR in its basic form; in Section 3 we will present additional design features that improve system performance and scalability.

Approximate index routing is illustrated in Figure 4. Summarization function *f* maps the index-value set into a new set of more compact format but with false positive probability. More specifically, *f* maps five index-values into a set of only three distinct values. "P2P" and "Network" are mapped into the same value 1. Therefore, at node *C*, the query for "P2P" is forwarded to node *E*, which in fact does not have "P2P" but has "Network".

### Bloom Filter as approximation index

AIR routing tables are implemented using Bloom Filters [1] . Bloom Filter is a randomized data structure for concisely representing a set in order to support approximate membership queries. It provides an attractive opportunity for tradeoff the space efficiency and accuracy of the representation.

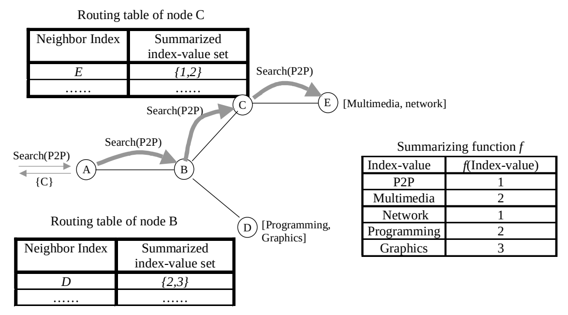


Figure 4: *Example of approximate index routing.*

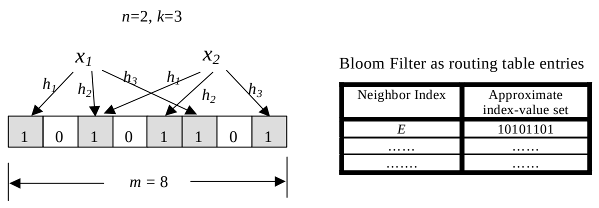


Figure 5: *Bloom Filter used in approximate index routing table.*

The principle of Bloom Filter is as follows:

To represent a set *A* of *n* elements, a Bloom Filter uses a vector *v* of *m* bits, all initially set to 0, and *k* independent hash functions, $h_1, h_2, \dots, h_k$, each with range $\{1, \dots, m\}$. For each element in *A*, the bits at positions $h_1(a), h_2(a), \dots, h_k(a)$ in *v* are set to 1. Bloom Filter answers a query for element *b* by checking the bits at positions $h_1(b), h_2(b), \dots, h_k(b)$. If any of them is 0, *b* is certainly not in the set *A*. Otherwise Bloom Filter gives the positive answer that *b* is in the set although there is a certain probability that it is not. This is called a "false positive" answer. The probability for the Bloom Filter to generate false positive answer is

$$\alpha = (1 - (1 - \frac{1}{m})^{nk})^k \approx (1 - e^{-\frac{kn}{m}})^k$$

$\alpha$ is minimized for $k = (m/n)\ln 2$ hash functions, in which case the right side of the above formula equals $(1/2)^k$.

A Bloom Filter example is shown in Figure 5. Using 8 bits and 3 hash functions to represent 2 index-values, the probability that the Bloom Filter gives a false positive answer is around 0.167. If this Bloom Filter is used in the AIR routing table of node *C* in Figure 4, for representing the two index-values at node *E*, the probability for node *C* to mistakenly forward an arbitrary query message to node *E* is 0.167. Dedicating more memory space in the routing tables can easily reduce such false positive network traffic. For example, using 16 bits and still 3 hash functions, the false positive rate drops to 0.033. Therefore, Bloom Filter offers a very flexible opportunity for AIR to tradeoff memory space usage and network bandwidth consumption. As memory space is the most constrained resource in content-based routing, we expect the price of a little extra network traffic may well be worthwhile to reduce the necessary routing table size.

## 2.3 AIR construction and maintenance

### Bootstrap and node join

The tree topology of AIR overlay network can be directly constructed or extracted from an underlying mesh. For simplicity, we only discuss the tree topology in this paper.

To join the network, the new node chooses only one existing node to be its neighbor in the acyclic graph. As in other P2P systems, a new AIR node initially knows about one or more nodes already in the system by some external means. Typical methods include using expanding ring IP multicast, configuration by system administrator through outside channels, or resolving bootstrapping node IPs associated with a well-known DNS domain. In choosing a node to connect with, new node may take many factors into consideration, such as geographic location, content locality, node capabilities and load condition etc. Some factors have impact on AIR performance and will be discussed further in Section 3. Others factors are orthogonal to AIR design and, therefore, we leave them as separate component to be integrated into AIR systems.

After joining the network, the new node advertises its local index values to its neighbor and builds its own routing table by learning abou objects in the system from its neighbor. It also starts to actively monitor advertisements from other nodes, to populate its routing tables.

### Node departure and failure

To address the lack of redundancy in the tree-topology overlay network, each AIR node selects one neighbor to be its "successor node". When a node wants to leave the network, it first notifies its successor. The successor node then copies the routing tables from the leaving node and connects with the node's other neighbors as its new neighbors. After these operations are done, the node leaves. The successor node uses the inherited routing tables to route for the new neighbors. If a node fails abruptly, its successor node still connects with the other neighbors and then copies the routing tables from these new neighbors.

The process is illustrated in Figure 6. The original topology is shown in Figure 6(a). Then, in (b), node *A* leaves and its successor node *B* connects with *A*'s other neighbors. From the figure, we can see that the successor node not only keeps the topology a connected acyclic graph, it also guarantees that the same relative positions between the nodes is maintained, i.e. the objects remain at the same directions to each node, so that the routing tables are still valid. In contrast, another mechanism in (c) also repairs the network to an acyclic graph, but the relative position between nodes is changed. Both node *C* and *E* have to modify their routing tables to reflect the change, which is a potentially expensive operation.

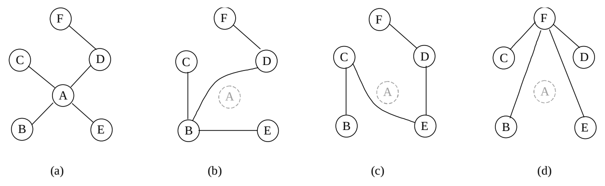


Figure 6: *AIR node departure.*

The AIR nodes at the edge of the network, i.e. the leave nodes in the tree topology do not have successor nodes. An inner AIR node chooses a neighbor with high capability and low connectivity load as its successor, so that the neighbor is able to accommodate the new connectivity when needed. However, a neighbor may refuse to be the node's successor if itself is under heavy load. If a node has no neighbor that is willing to be its successor, it may look further and choose a capable non-neighbor node. For example, as shown in Figure 6(d), node *F* becomes node *A*'s successor. In this case, there must be one neighbor *D* that is originally on the path from the node *A* to the successor *F*. *A* tells *F* about *D* so that when *A* leaves, *F* connects to all *A*'s other neighbors except *D*. The ability to choose a successor node multiple hops away helps to achieve global load balance. However, the further distance between a node and its successor, the larger "disruption" is brought to the network, as the nodes that were previously far away in the overlay are now close and vice versa. Such change may be unfavorable in preserving the properties built at the time of network construction, such as geographic or content locality. On the other hand, the successor node mechanism can be used for adaptation purpose. When the network topology is sub-optimal, an AIR node actively adjusts the connectivity by leaving the network and connecting its neighbors to a more appropriate area.

To address the possibility of simultaneous failure of a node and its successor, each node may choose a sorted list of successor nodes. A node in this list serves the successor responsibility if and only if all the nodes before it are gone. The node and the successor node(s) periodically exchange their neighbor information.

### Object insertion and deletion

The routing tables are constructed and populated by listening to advertisement message from neighbors, in a similar fashion to IP routing protocols, such as OSPF. When a new object is inserted at an AIR node, the node advertises its index-values by sending "new object message" to all its neighbors, and the neighbors forward the message on to all other nodes. For network efficiency, the "new object message" are batched or piggybacked with other messages. During the delayed period, some queries may get false negative results. Therefore, the frequency of transmission of the message involves a trade-off between routing accuracy and efficiency. Similarly, when an object is deleted, a node sends out an "object deleted message". A node receiving the message forwards it to a neighbor only when it knows that the object is neither stored on the other directions.

Inserting objects into a Bloom Filter is easy: hash the object index value *k* times and set the bits to 1. Unfortunately, one cannot perform a deletion by reversing the process. If we set the corresponding bits to 0, we may be clearing a location that is hashed by some other objects in the set. In this case, the Bloom Filter no longer correctly represents all the objects. To avoid this problem, [10] introduces the idea of *counting Bloom Filter*, in which each entry of Bloom Filter is not a single bit but a small counter, to keep track of how many objects have set the bit. Here we propose another method that uses less space for small number of deletions. Each routing table is associated with a "deletion list" of index-values that have been recently deleted. A query is forwarded to a neighbor only when the queried index-value is not in the "deletion list" and results in a hit in the Bloom Filter. The deletion list can also be used to cache index-values that result in the most frequent false positive routing on a direction, based on the feedback from the nodes in that direction.

## 2.4 Supporting unicast search

As discussed before, the basic AIR routing semantic supports multicast search, meaning it is capable of locating all replicas of one object or all objects matching a query message. The strategy used is parallel search in which a query message is forwarded to the subset of neighbors who satisfied the routing table lookup. The result list of locations can be augmented with a ranking system to help user finally choose one location to perform the actual data retrieval. Distributed search system is one typical application.

Alternatively, in some applications, it is desirable for the routing layer to locate only one copy of the object queried. For example, in a distributed file sharing system, a popular file is cached over a fair fraction of the network. Multicasting a query message for this file to all its replicas while only one copy is wanted can bring high, unnecessary routing overhead on the system, not to mention that queries for such popular files are likely to be common.

For this purpose, AIR supports another interface: *unicastSearch(index-value)*, which returns only one location for the matching object, if it exists in the system. The routing mechanism can simply be changed to sequential routing or random walk, e.g. forwarding to one candidate neighbor at a time, until an object is located or a negative result is returned. However, the random location thus identified may not be a good one in terms of application preference. For example, it is often desirable to locate one copy of the file that is "close" to the querying user, in terms of latency or/and access bandwidth. Study in [25] showed that in a distributed P2P file sharing system, there are three large classes of latencies that a peer interacts with: latencies to peers on the same part of the continent, latencies to peers on the opposite part of the continent, and latencies to trans-oceanic peers. Latency is a very important consideration when selecting amongst multiple peers in this application, because of the well-known feature of TCP congestion control that discriminates against flows with large rotund-trip times, and the large size of the files exchanged (usually in the order of 2-4 MB).

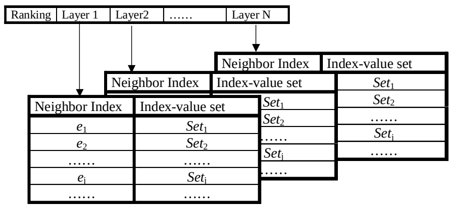


Figure 7: *Layered AIR routing table.*

To enable more intelligent decision-making, an AIR routing table is divided into several layers, each layer associated with a rank range, as shown in Figure 7. Simple information, such as location, of an object is encoded in the index-value advertisements. The index-value is then ranked based on such information or other node statistics, and inserted into the corresponding layer. When a query message is received, the top-ranked table layer is consulted first. If all answers are negative, the next layer is checked. Otherwise, the message is forwarded to one direction that is indicated to have the object at that layer. Note that due to the false positive probability of approximate index, the message may have been sent to a direction that does not lead to any matching object. In this case, a negative acknowledgement is sent back by the first node that detects the mistake. The initiating node then tries the next best-ranked direction that is said to have the object. If the false positive ratio is high, two directions can be tried simultaneously. As the process is in fact a distributed breadth-first search, *unicastSearch* is expected to return a close-to-optimal copy of the object in the system.

While we believe that a ranking module can be plugged into AIR system rather easily, we leave the evaluation of different strategies as future work. For simplicity, in the rest of this paper, we will concentrate on the original multicast-search scheme.

# 3. AIR improvements and evaluation

In this section, we describe AIR design in more detail and present analytical and simulation-based evaluations of AIR performance, in terms of balancing the tradeoff between memory consumption and network traffic load to achieve global system scalability. Furthermore, we show that AIR is able to exploit optimization opportunities inherent in P2P applications, such as resource heterogeneity and content locality.

## 3.1 Independent false positives

The first observation is that if every AIR node indexes the objects in the same way, there is high level of information redundancy between the routing tables. An example is shown in Figure 8. Assume that the index-value of objects *i* and *j* happen to conflict for all the hashing functions used in Bloom Filter $BF_1$. In this case, if either *i* and *j* is stored on one direction, a node using $BF_1$ in its routing table always forwards queries for the other object to that direction too, even if the other object was not there. Figure 8(a) shows that when every AIR node uses $BF_1$, a query message for object *i* is forwarded all the way to the nodes with object *j*, resulting in high false positive routing. In contrast, if the AIR nodes use different Bloom Filters, as shown in (b), the false positive routing is stopped quickly. The reason for this is that it is unlikely that object *i* and *j* conflict again in the Bloom Filters used by the other nodes on the path. If the false positive results of these routing tables are independent, the probability such false positive routing continues drops exponentially at each step the message is forwarded. Specifically, assuming that the queried object is not stored in the system, we have

$$\begin{aligned}&Expectation(\text{false positive routing path length})\\&=\alpha+\alpha^2+\alpha^3+\ldots=\frac{\alpha}{1-\alpha}\end{aligned}\tag{1}$$

Where $\alpha$ is the false positive probability of one Bloom Filter. Immediately following (1), we have

$$\begin{aligned}&Expectation(\text{false positive routing traffic})\\&=\alpha n_1+\alpha^2 n_2+\alpha^3 n_3+\ldots=\sum_{i=1}^{n-1}\alpha^i n_i\end{aligned}\tag{2}$$

Where $n_i$ is the number of nodes at distance *i* from the initiating node. In contrast, for the case of all nodes using the same Bloom Filters, we have

$$\begin{aligned}&Expectation(\text{false positive routing path length})\\&=\alpha+\alpha+\alpha+\ldots=\alpha APL(n)\end{aligned}\tag{3}$$

Where *APL*(*n*) is the average path length in the overlay network, and

$$\begin{aligned}&Expectation(\text{false positive routing traffic})\\&=\alpha n_1+\alpha n_2+\alpha n_3+\ldots=\alpha(n-1)\end{aligned}\tag{4}$$

Clearly, the value of (3) and (4) are much larger than that of (1) and (2), especially for large network size *n*. Therefore, using different Bloom Filters in AIR greatly reduces the false positive routing compared to using the same ones. For the case that there is matching object(s) stored in the system, AIR guarantees to route the message to the node that has the object. The total false positive network traffic can be seen as the sum of those starting from each node on the routing path, and the results are similar.

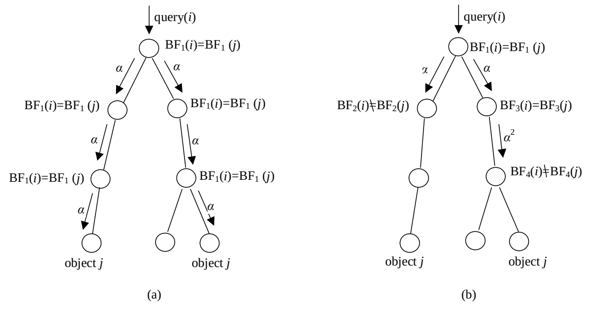


Figure 8: *AIR with same or different Bloom Filters*

The idea of using different Bloom Filters at different nodes shares the same intuition with the DHT systems. In both cases, the nodes are designed to take different "views" of the system, so that at each step a message is forwarded, it benefits from the new knowledge and becomes more (accurately) informed about its destination.

**Bloom Filter configurations**

We present AIR performance using two configurations for Bloom filters: the number of bits for each index-value (*m*/*n*) being 4 and 8, and we always use 4 hash functions. We present the result of these two configurations because their tradeoff between memory space consumption and network overhead generated are expected to be of most practical interest. As will be shown later, using 8 bits per index-value, AIR achieves so good accuracy that we expect further improvement is not worth the additional space required. Using 4 bits per index-value generates more network overhead but may still be acceptable. More interesting results of combining these two configurations will be also discussed.

The hash functions are built by first concatenating local IP address with the index-value, then calculating the MD5 signature [18] of the result string, which yields 128 bits. We divide the 128 bits into four 32-bit words, and take the modulus of each 32-bit word by the bit vector size *m* as the four hash values for the index. MD5 is a cryptographic message digest algorithm that hashes arbitrary length strings to 128 bits. We select it because of its well-known properties and relatively fast implementation. The Bloom Filters thus constructed use different hash functions at each node and result in independent false positive probabilities. This concept will serve as the basis for our analysis.

## 3.2 Basic AIR performance

In this section, we present performance of basic AIR design, in which AIR nodes use different Bloom Filters but with the same configuration. Because AIR overlay topology is configurable based on application-specific features, we experiment with two highly different tree topologies and study their impact on AIR performance. First, we generate the tree topology using the network construction process described in Section 2: each time when a new node joins the network, it uniformly randomly selects an existing node to be its neighbor. We call such a tree an evolvement tree. We then generate a uniform random tree topology, using the random walk process proposed by Broder in [1] . The random tree has a uniform probability to be any one of all the possible tree structures connecting the *n* nodes in the system. The degree distributions of these topologies are compared in Figure 9, together with the power law distribution, which is found to prevail in the degree distribution of Internet topologies. Figure 9 shows that an evolvement tree has a higher fraction of high-degree nodes than the power law distribution, while most of the nodes in the random trees have very low degrees. Therefore, the evolvement tree has a relatively dense shape while the uniform tree has a sparser one. In this paper, we will focus on analysis in the evolvement tree topology, as it is the more realistic topology for a P2P overlay network.

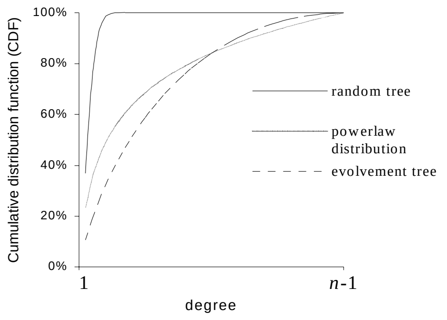


Figure 9: *Comparison of three degree distributions.*

We compare the network traffic generated in AIR with a conjectured “accurate routing” scenario, in which all the routing tables are perfectly accurate and the query messages are forwarded toward and only toward the nodes that have the queried objects. Accurate routing achieves optimal network efficiency in a given AIR overlay network. However, we expect such high efficiency comes at the cost of larger memory space requirement and/or more complicated routing table lookup operations.

AIR performance is evaluated in terms of total network traffic generated for routing a query message.

Figure 10 shows the simulation results of average number of hops a query message propagates when routed to a node that has the object. For accurate routing, this number equals to the path length from the initiating node to the destination node (there is only one such path in the acyclic overlay network). For AIR, it includes the correct routing on the path, and some extraneous routing off the path due to false positive index results.

As shown in Figure 10, the extraneous network traffic in approximate routing is quite low in both evolvement tree and random tree topology. Especially, with the configuration of 8 bits per index-value, AIR performance is very close to that of accurate routing. Because the evolvement tree has a denser structure, its average distance for both accurate and approximate routing is less than in the random tree. However, the fraction of false routing in total network traffic is lower in the random tree than in the evolvement tree. This can be explained by formula (2):

$$Expectation(\text{false positive routing traffic})$$
$$= \alpha n_1 + \alpha^2 n_2 + \alpha^3 n_3 + \ldots = \sum_{i=1}^{n-1} \alpha^i n_i$$

Which indicates that in a dense graph with many high degree nodes, $n_i$ is large for small *i*, and the expectation of false positive network load would be relatively high.

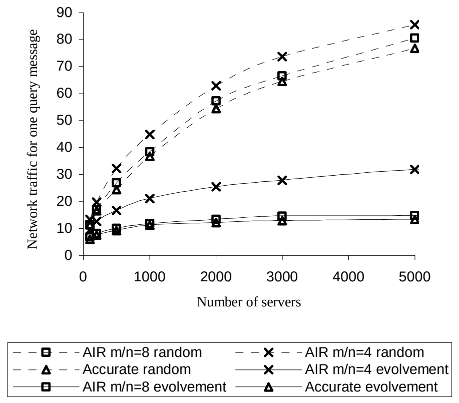


Figure 10: *Simulation results of AIR performance in routing query message to matching objects in system.*

In appendix, we analyze AIR and accurate routing in evolvement tree in more detail. We show that the average path length in evolvement tree is O(log*n*), i.e. a query message arrives at the destination node in O(log*n*) steps. Based on the recursion formulas in appendix, we plot the AIR overhead, in terms of percentage compared with total traffic in accurate routing, for network of larger scale in Figure 11. In a network of 10 million nodes, using 8 bits for each index-value, the approximate routing overhead is as low as 20%. Using 4 bits for each index-value, the overhead is 3 times of the accurate routing. In practice, such overhead may still be acceptable, especially if compared with the network bandwidth consumed in downloading the object located.

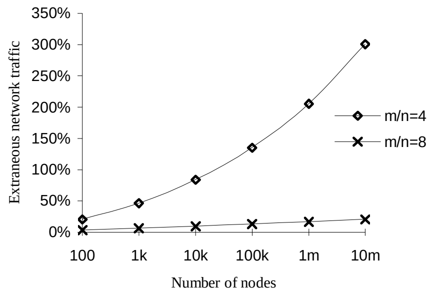


Figure 11: *Analytical results of AIR overhead in routing query message to matching objects in the system.*

If the P2P system does not have any object matching a query message, we study the average number of hops such message propagates in AIR, due to the false positive index results. Table 1 shows that on average, AIR forwards the query for no more than one hop. In appendix, we prove that this number converges to $2\alpha/(1-2\alpha)$ in the evolvement tree, where $\alpha$ is the false positive ratio of one Bloom Filter. This limit equals 0.543 for $m/n$=4 and 0.056 for $m/n$=8. In comparison, recall that the message would be routed *O(logn)* steps in DHT systems, and flooded over the network in Gnutella.

Table 1: *AIR performance for routing query message that has no matching object in the system.*

| #nodes | Average number of hops a non-hit query message is forwarded | | | |
|---|---|---|---|---|
| | Random tree | | Evolvement tree | |
| | m/n=4 | m/n=8 | m/n=4 | m/n=8 |
| 100 | 0.400 | 0.047 | 0.523 | 0.052 |
| 200 | 0.412 | 0.049 | 0.525 | 0.053 |
| 500 | 0.422 | 0.049 | 0.530 | 0.053 |
| 1000 | 0.442 | 0.050 | 0.533 | 0.053 |
| 2000 | 0.445 | 0.052 | 0.541 | 0.054 |
| 5000 | 0.447 | 0.052 | 0.538 | 0.054 |

## 3.3 Exploiting Heterogeneity

So far, we have made the implicit assumption that the responsibility across AIR nodes is uniform, and that all nodes participate and contribute equally in AIR routing. Measurements in [20] indicate that the set of hosts participating in real P2P systems such as Napster [19] and Gnutella is heterogeneous with respect to many characteristics: Internet connection speeds, latencies, lifetimes, and shared data. In this section we study how to actively exploit node heterogeneity in AIR.

The strategy is to associate the nodes' contributions to AIR to their capabilities. For example, the nodes with more available memory space maintain their routing tables with higher accuracy; those with high bandwidth, low latency network access take more AIR neighbors. At the same time, the requirement for less capable nodes is reduced.

AIR nodes are divided into three categories: stub nodes, routers and super routers. Stub nodes are to represent the most unstable and incapable nodes in a P2P system. Due to their short lifetime in the system or/and lack of resource, stub nodes do not maintain AIR routing tables. Instead, they unconditionally forward all query messages received to their AIR neighbors. Routers and super routers are the more reliable and capable participants that take the responsibility of AIR routing. Super nodes are like the "hub nodes" in Gnutella or KaZaa that are highly stable and with abundant resources. In their routing table configurations, we assume that super routers use 8 bits for each index-value while normal routers use 4 bits.

We experiment with two proportions of the three categories as shown in Table 2. They are set to model the distribution found in [25] of Napster and Gnutella users with network connections of less than 300Kbps, between 300Kbps and 3Mbps, and beyond 3Mbps. During the process of network construction, we assume super routers are the nodes that join the AIR network first, then the normal routers, and finally the stub nodes. This sequence is consistent with the stability characteristics, and result in connectivity distribution that naturally biases toward nodes with high capabilities. Each new AIR node still randomly selects an existing node to be its neighbor. This is because the metric used in network construction can be orthogonal to the resource characteristics.

Table 2: *Distribution of heterogeneity in AIR.*

| | Stub nodes | Routers | Super routers |
|---|---|---|---|
| Hetero_2_1_1 | 50% | 25% | 25% |
| Hetero_5_4_1 | 50% | 40% | 10% |

The performance of heterogeneous configurations is presented in Table 2, in comparison with the homogeneous configurations described in Section 2. Note that in terms of total memory usage in overall AIR system, both heterogeneous configurations require less amount of memory space than the homogeneous configurations.

Table 3 shows that both heterogeneous configurations greatly improved AIR accuracy in routing a query message to its destination node, compared to the homogeneous 4 bits per index-value configuration. In fact, the performance of hetero_2_1_1 even approaches the homogeneous 8 bits per index-value case, in which every node is configured as a super router. Further reducing memory requirements, hetero_5_4_1 configuration can still keep false positive network overhead below 25%. The reason for this is that in the heterogeneous settings, the nodes with low "filtering power" have also low network connectivity. The false

Table 3: *AIR performance with homogeneous and heterogeneous configurations*

| # nodes | Extraneous network traffic in routing query message to destination node (in percentage compared to accurate routing) | | | | Average number of hops a query message is transmitted when the queried object is not stored in the system | | | |
|---|---|---|---|---|---|---|---|---|
| | Homogeneous configurations | | Hetero_2_1_1 | Hetero_5_4_1 | Homogeneous configurations | | Hetero_2_1_1 | Hetero_5_4_1 |
| | m/n=4 | m/n=8 | | | m/n=4 | m/n=8 | | |
| 100 | 56.53% | 10.05% | 9.84% | 21.05% | 0.523 | 0.052 | 0.667 | 0.820 |
| 200 | 66.42% | 9.93% | 9.51% | 20.89% | 0.525 | 0.053 | 0.668 | 0.844 |
| 500 | 80.46% | 10.76% | 10.13% | 21.55% | 0.530 | 0.053 | 0.674 | 0.853 |
| 1000 | 88.28% | 11.97% | 11.27% | 23.00% | 0.533 | 0.053 | 0.686 | 0.847 |
| 2000 | 107.88% | 12.53% | 12.16% | 23.64% | 0.541 | 0.054 | 0.682 | 0.843 |
| 5000 | 139.15% | 12.97% | 12.65% | 24.02% | 0.538 | 0.054 | 0.690 | 0.846 |

positive decisions these nodes make generate relatively low real network traffic. An extreme case is that for a stub node that has only one degree. When the queried object is stored anywhere in the system other than the stub node itself, the unconditional forwarding by such a stub node is in fact usually a correct operation. The situation is different if the queried object is not in the system. In this case, the unconditional forwarding is always a wrong one. However, the network traffic generated is still quite low and the queries can still be dropped off within less than one step.

Exploiting heterogeneity in node characteristics is highly effective in improving scalability of the AIR system. Now, the efficiency of AIR routing no longer relies on the premise that *every* participant exhibits satisfying performance. Instead, it is enough if *some* nodes in the system are capable to do so. The system is now able to accommodate a more diversified set of nodes, especially those that are volatile and less cooperative. It is especially promising that even when half of the nodes keep no routing information at all, AIR still achieves fairly high accuracy. The technique to exploit heterogeneity is also easy to deploy. The heterogeneous configurations do not have to fall into an explicit hierarchy. Each AIR node can configure its Bloom Filter to match its characteristic with high flexibility.

## 3.4 Data grouping

Up to now, each AIR node maintains global index for all objects in the system. They do this more for "common good" than for their self-interest. It has been observed that users in P2P systems have highly focused interests [30] . For example, a user may care for particular genre of music or certain field of scientific study. To exploit such interest locality, we propose the technique of data grouping.

The logical space of index-values is partitioned into *g* groups. An object is said to belong to one group if one of its index-value falls into the group. The principle of the partition is to group "similar" objects together, i.e. the objects that belong to the same group have higher probability to be queried or/and stored together than the objects in different groups.

An AIR node is said to have interest in a group if it has objects in that group that it wants to share, or that it wants to query about objects in that group. For each group, all the interested nodes organize into an acyclic overlay network in the same way as in the basic AIR. We call such an overlay a group-tree. When receiving a query message, the AIR node first determines the group the queried index-value belongs to, and then routes the message in the corresponding group-tree.

In simulation, we divide index-values into 20 groups. Each AIR node randomly joins *k* of these groups, based on each group's popularity. The group popularity is set to follow Zipf distribution [3] , i.e. the probability that a node joins group *i* is proportional to the ranking of group *i* in terms of its popularity: $P_i \sim i^{-\alpha}$. $\alpha$ is set to 1, based on results in [31] . The implication of this distribution is that a few groups are very popular while most groups have low popularity. Figure 12 shows the real group sizes, in terms of percentage of nodes it has, in descending order. Note that the distribution is not always consistent with the popularity distribution; instead, it is affected by *k*, the number of groups each node joins. The broader a node's interest is, the more evenly distributed the group sizes. The average routing table size after grouping is shown in Figure 13. For comparison, we also plot the results for uniform group popularity case. It is clear that the more focused a node's interest is, the smaller routing table is needed. The difference between the two curves is because with group popularity as Zipf distribution, a node has higher probability to join a popular group, which also tends to have more members. Study in [5] , [30] showed that in a P2P file sharing system, there often exist small subsets of users that share the same interests. In a system of thousands of users, a high fraction of a user's queries can be answered by asking only ten other users. Therefore, in reality, we expect each AIR node joins a small number of groups, and the routing table sizes are significantly reduced compared to without data grouping. Furthermore, because the query messages now propagate only between a subset of nodes, the average routing path length is reduced. The network traffic load is the same as querying in a smaller network with the size of the group.

Data grouping also offers flexible optimization opportunities. The overlay network topology and configuration strategy for each group-tree can be finely customized based on the special features in the data group. The overall AIR overlay network with multiple embedded group-trees is also more robust and resilient than a single tree topology.

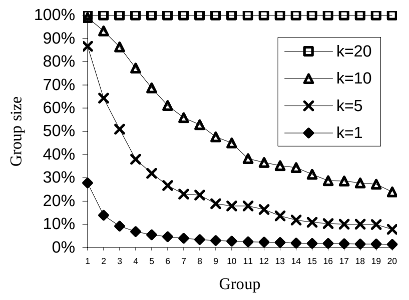


Figure 12: *Group sizes based on Zipf group popularity.*

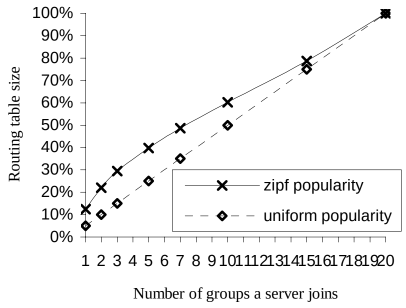


Figure 13: *Routing table sizes with data grouping.*

## 4. Applicability analysis

### Scalability

Scalability of AIR system has two folds, the routing table space consumption and the network traffic load. We have shown that AIR routing can achieve close-to-optimal accuracy. In this section we analyze the AIR scalability based on the memory space requirement and management cost.

AIR indexes on the index-values of objects. For applications such as keyword-based search in a P2P file sharing system, the number of index-values can be higher than the object quantity, while when using AIR to support group communication, many objects share the same index-value. In our analysis, we simply assume that the number of distinct index-values in the system to be roughly equal to the total number of objects.

We consider a P2P system with 100 million objects. Assume that each user is interested in 10% of all the objects, and the content popularity follows Zipf distribution. Figure 13 shows that the average AIR routing table indexes about 20% of all objects. Using 8 bits per object configuration, the Bloom Filters in a routing table need 20%×100×8/8 =20 MB space. The routing table may also contain other information such as ranking, layering and cached deletion list. We assume that these data adds to no more than 10 MB. At the time of writing, 30 MB is a very reasonable memory requirement for an application software running on a normal PC. We expect this much memory space to be easily affordable by an AIR node, especially the routers and super routers. (The algorithm of AIR does not require routing tables to be in memory. We do not consider the possibility of using disk space to avoid degradation of AIR performance.)

As stated in Section 2, object insertion and deletion trigger advertisement messages that may propagate to the entire network. Assume that in a dynamic P2P system, 10% of all content change every day, and each update message contains 1000 bits. The network traffic of these update messages is 20%×100×10%/(24×3600)=23.15 Kbps. Note that the update messages are only exchanged between AIR routers and super routers, which are supposed to have network bandwidth higher than 300 Kbps. Therefore, the background traffic consumes less than 7.7% of the total network bandwidth available, which should be a reasonable management overhead.

From these analyses, we expect AIR to be able to support P2P systems of hundreds of millions of objects.

### Potential applications

One of the motivations for AIR work is to provide scalable object location scheme for P2P applications that cannot be directly supported by DHT systems. A fundamental difference between DHT and AIR functionality is that AIR inherently supports multiple object location. This can be very useful in supporting application level group communication. Furthermore, because AIR can support large number of index-values, the communication can also be based on finer criteria than explicitly established groups. A typical application is distributed content-based publish-subscribe system [10] . In such system, a user subscribes for certain kind of events that are going to happen in the future. Subscriptions are submitted to a server (node) in an overlay network. Events are published to any of the servers in the system and AIR can be used to route the events to all the nodes that have matching subscriptions.

AIR can also be used in implementing a write-able file sharing system. Update is possible because AIR can locate all replicas of a file and maintain consistency between multiple copies.

Another application for AIR is distributed search engine. Bloom Filters offers space saving and faster membership check than text-based operations. However, an open question is how to efficiently support Boolean queries on distributed index structures.

## 5. Related work

There have been many recent works on improving or extending one of the two major established P2P system models: the Gnutella-like unstructured systems and the highly structured DHT systems.

[5] , [30] proposed to improve scalability of Gnutella-like systems by exploiting *interest-based locality*. The basic assumption is that in a P2P file sharing system, if a peer has content that matched a previous query, it is very likely that it also has content that will match other queries from the same peer. Such peers are said to share the same interest. The approach is to build additional overlay structure on top of the Gnutella network, and direct search messages to first propagate within this overlay. Only when the overlay fails to locate an object will the message be flooded over the network. Results show that the approach can decrease the amount of traffic load in the system and the time to locate content. However, because of their reliance on the underlying Gnutella scheme, a query message is not guaranteed to find an object stored in the system.

To actively exploit the heterogeneous characteristics across the peers in a P2P system [27] , Lv et. al [17] proposed a flow control method that matches the skewed network load, generated by the routing algorithms they used, with heterogeneous peer capabilities. [14] introduced a two-level hierarchy architecture that groups peers into clusters. The "delegate node" in each cluster takes more routing responsibility and serves as a central directory server for the cluster. These schemes do not guarantee finding an existing object either.

The DHT functionality has proved to be a useful substrate for large distributed systems; a number of projects are proposing to build Internet-scale facilities layered above DHTs. [8] and [25] are two read-only distributed storage systems built on top of Chord [32] and Pastry [9] . To support data replication, caching, and load balance, both systems modified the routing algorithm and semantic of the underlying DHT overlay. Another persistent storage utility, OceanStore [15] , provides data privacy, allows client updates, and guarantees durable storage. However, these features come at the price of high complexity, and it is assumed that the core system will be maintained by commercial providers. DHT routing has also been used to implement application-layer multicast [21] [34] and event notification services [5] [26] . In these works, DHT routing is used to locate the rendezvous node for a communication group, and a separate multicast tree has to be built for real data dissemination. Unlike the unstructured systems, the tight coupling of data content and their location in DHT systems has relatively restricted their potential in exploiting content locality and resource heterogeneity.

In this paper, we improve the efficiency of query routing in a simple, unstructured P2P system. The routing scheme is effective in the sense that all matching objects are guaranteed to be located. The idea of content-based routing has been used in distributed publish-subscribe systems [6] and intra-agent communications [29] . The typical system architecture and content format in these applications are usually different from a P2P system and different methodology is used.

In AIR implementation, we used the technique of Bloom Filter to balance the tradeoff between routing accuracy and index scalability. Bloom Filter has been used in Web caching [11] , resource routing [23] , and free text search [22] . We are different from these works in that AIR is achieved through cooperation of independent Bloom Filters in an overlay network. To our best knowledge, no work has studied the performance and properties of such systems.

## 6. Conclusion and future work

In this paper, we proposed the approach of content-based routing for query resolution in a P2P system, and presented Approximate Index Routing as one implementation.

We explored the tradeoff in routing based on approximate index, and found that cooperation between nodes with different approximations greatly improves routing accuracy and efficiency. At the same time, AIR is highly effective in that it guarantees to find all matching objects in the system, and stops quickly if such object does not exist. Due to its simple and flexible system structure, AIR is able to take advantage of application specific properties such as resource heterogeneity and content locality. In all, analysis shows that AIR can scale to P2P systems of hundreds of millions of objects, and provides a promising routing solution for many potential applications.

Much future work remains. We plan to deploy approximate index routing in a real P2P system, and further analyze its dynamic behavior. We also plan to investigate the optimal configurations of AIR for a given application background. An intriguing question is how to implement AIR in more flexible overlay topologies, such as mesh networks.

## References

[1] B. Bloom, "Space/time trade-offs in hash coding with allowable errors". In Commun. ACM, vol. 13 no. 7, pp. 422-426, July 1970.

[2] A. Broder. "Generating random spanning trees". In IEEE 30th Annual Symposium on Foundations of Computer Science, pages 442--447. IEEE, 1989

[3] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, Scott Shenker (2000), "Web caching and Zipf-like distributions: evidence and implications", Proceedings of INFOCOM'99 (IEEE Press).

[4] A. Broder, M. Mitzenmacher. "Network applications of bloom filters: A survey", In Allerton, 2002

[5] L. F. Cabrera, M. B. Jones, and M. Theimer, "Herald: Achieving a global event notification service", In HotOS VIII, May 2001.

[6] A. Carzaniga, D. Rosenblum, and A. Wolf. "Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service". In Proceedings of the 19th Annual Symposium on Principles of Distributed Computing (PODC 2000.

[7] E. Cohen, A. Fiat, and H. Kaplan, "A case for associative peer-to-peer overlays". In Proc. Of Workshop on Hot Topics in Networks, 2002.

[8] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, I.Stoica, "Wide-area cooperative storage with CFS". In proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)

[9] P. Druschel, and A. Rowstron, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middle-ware 2001) Nov 2001.

[10] P. T. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. "The Many Faces of Publish/Subscribe". Technical report, EPFL, Lausanne, Switzerland, 2001

[11] L. Fan, P.Cao, J. Almeida, and A.Z. Broder. "Summary cache: a scalable wide-area Web cache sharing protocol". In IEEE/ACM Transactions on Networking, 8(3): 281-293, 2000

[12] Gnutella, http://gnutella.wego.com.

[13] KaZaa, http://www.kazaa.com.

[14] B. Krishnamurthy, J. Wang, and Y. Xie, "Early Measurements of a Clusterbased Architecture for P2P Systems," in Proceedings of ACM Sigcomm Internet Measurement Workshop, November 2001.

[15] J. Kubiatowicz, D. Bindel, P. Eaton, Y. Chen, D. Geels, R. Gummadi, S. Rhea, W. Weimer, C. Wells, H. Weatherspoon, and B. Zhao. "OceanStore: An architecture for global-scale persistent storage". In ACM SIGPLAN Notices, 35(11): 190-201. November 2001.

[16] Q. Lv, P. Cao, K. Li, and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," In Proceedings Of ACM International Conference on Supper Computing(ICS), 2002.

[17] Q, Lv, S. Ratnasamy, S. Shenker, "Can heterogeneity make Gnutella scalable", In First International Workshop on Peer-to-Peer Systems (IPTPS) 2002, Cambridge, MA, USA, March 2002.

[18] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, "Handbook of applied cryptography". CRC Press, 1997

[19] Napster, http://www.napster.com

[20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, "A scalable Content Addressable Network". In Proc. ACM SIGCOMM, pp. 161-172.

[21] S. Ratnasamy, M. Handley, R. Karp, S. Shenker, "Application-level Multicast using Content-Addressable Networks". In Proceedings of NGC 2001

[22] P. Reynolds and A. Vahdat. "Efficient peer-to-peer keyword searching". Unpublished manuscript.

[23] S. C. Rhea and J. Kubiatowicz. "Probabilistic location and routing". In Proc. of INFOCOM 2002.

[24] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design". In IEEE Internet Computing Journal, vol. 6 no. 1, 2002.

[25] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility". In 18th ACM SOSP'01, Lake Louise, Alberta, Canada, October 2001.

[26] A.I.T. Rowstron, A.-M. KermalTec, M. Castro, and P. Druschel. "SCRIBE: The design of a large-scale event notification infrastructure". In Networked Group Communication, pages 3043, 2001.

[27] S. Saroiu, K. Gummadi, S. Gribble, "A measurement study of peer-to-peer file sharing systems". In proceedings of Multimedia Conferencing and Networking 2002.

[28] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy. "An Analysis of Internet Content Delivery Systems". In Proceedings of OSDI 2002, December 2002

[29] N. Skarmeas and K. Clark, "Content based routing as the basis for intra-agent communication", To appear in Intelligent Agents V, Springer Verlang.

[30] K, Sripanidkulchai, B, Maggs, H, Zhang, "Efficient content location using interest-based locality in peer-to-peer system", In Infocom 2003

[31] K, Sripanidkulchai, "The popularity of Gnutella queries and its implications on scalability." In O'Reilly's www.openp2p.com (Feb. 2001).

[32] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications." In Proceedings of the ACM SIGCOMM 2001

[33] B.Y. Zhao, J. Kubiatowicz, A. Joseph. "Tapestry: An infrastructure for fault-tolerant wide-are location and routing". Tech. R1ep. UCB/CSD-01-1141., University of California at Berkeley, Computer Science Department, 2001.

[34] S. Q. Zhuang, B. Y. Zhao, and A. D. Joseph. "Bayeux: An architecture for scalable and fault-

tolerant wide-area data dissemination”. In 11th ACM/IEEE NOSSDAV, New York, June 2001.

## Appendix A. Index routing on evolvement tree

This section presents analysis of network performance of index routing on the evolvement tree topology. AIR nodes are assumed to use different Bloom Filters but with same configuration. Therefore, the false positive result at each node is independent but the probability, denoted as $\alpha$, is the same across all nodes.

The proofs are by induction based on the process that the evolvement tree is constructed.

**Theorem 1.** A query that has no matching object in the system is called a *non-hit* query. The expected number of hops a non-hit query message propagates in AIR converges to $2\alpha/(1-2\alpha)$ with $\alpha<1/2$.

**Proof of Theorem 1:** Let $v_i$ be a node in the evolvement tree of $n$ nodes and $f_{vi}(n, \alpha)$ be the expectation of number of hops a non-hit query message initiating from $v_i$ propagates. Let $n_i$ be the number of nodes at distance $i$ from node $v_i$ in the evolvement tree. Because the false positive probability on each node is independent, we have

$$f_{v_i}(n,\alpha) = \sum_{i=1}^{n} n_i \alpha^i \tag{5}$$

When a new node $v_{n+1}$ joins the tree, it connects to any existing node with probability $1/n$. If it connects with $v_i$, a non-hit query message starting from $v_i$ has probability of $\alpha$ to traverse the edge $(v_i, v_{n+1})$; if $v_{n+1}$ connects with a node $v_j$ and $j \neq i$, a non-hit query message starting from $v_i$ has probability $\alpha^{dij+1}$ to traverse the edge $(v_j, v_{n+1})$, where $d_{ij}$ is the distance between node $v_i$ and $v_j$. Therefore,

$$\begin{aligned} f_{v_i}(n+1,\alpha) &= f_{v_i}(n,\alpha) + \frac{1}{n}\alpha + \frac{1}{n}\sum_{j\neq i}\alpha^{d_{ij}+1} \\ &= f_{v_i}(n,\alpha) + \frac{1}{n}\alpha + \frac{\alpha}{n}\sum_{k=1}^{n} n_k \alpha^k \\ &= f_{v_i}(n,\alpha) + \frac{\alpha}{n}(1 + f_{v_i}(n,\alpha)) \end{aligned} \tag{6}$$

A non-hit query message from node $v_{n+1}$ has probability $\alpha$ to propagate one step to its neighbor and then it can be seen as a non-hit query message initiating at its neighbor. Therefore,

$$f_{v_{n+1}}(n+1,\alpha) = \frac{\alpha}{n}\sum_{i=1}^{n}(1 + f_{v_i}(n,\alpha)) \tag{7}$$

Let $g_n(\alpha)$ be the sum of expected network traffic for non-hit query messages initiating from each of the $n$ nodes, i.e.

$$g_n(\alpha) = \sum_{i=1}^{n} f_{v_i}(n,\alpha) \tag{8}$$

Summing (6) for all $n$ nodes and (7) we have

$$\begin{aligned} g_{n+1}(\alpha) &= g_n(\alpha) + 2\alpha + \frac{2\alpha}{n} g_n(\alpha) \\ &= (1 + \frac{2\alpha}{n}) g_n(\alpha) + 2\alpha \end{aligned} \tag{9}$$

Now let $h_n(\alpha)$ be the average expected network traffic for non-hit query message initiating from a random node,

$$h_n(\alpha) = \frac{g_n(\alpha)}{n} \tag{10}$$

Substituting (10) to (9) we have

$$h_{n+1}(\alpha) = 2\alpha + 2\alpha h_n(\alpha) \tag{11}$$

It is clear that if $\alpha$<1/2, $h_n(\alpha)$ converges to $2\alpha/(1\text{-}2\alpha)$ and does not converge for $\alpha \geq 1/2$. □

In practice, recall from Section 2 that the false positive rate of a Bloom Filter is minimized when using $k = (m/n)\ln 2$ hash functions, in which case

$$\alpha \approx (\frac{1}{2})^k < \frac{1}{2}, \quad \text{for } k > 1$$

**Theorem 2.** If a query has a matching object stored in the system, the total network traffic in AIR to route the query message to a node with the object is O($n^\alpha$) hops.

**Proof of Theorem 2:** Let $f_{vi,vj}(n, \alpha)$ be the total network traffic generated in AIR to route a message from node $v_i$ to node $v_j$. With accurate routing, $f_{vi,vj}(n, \alpha)$ would equal to the number of hops on the path in the tree connecting $v_i$ and $v_j$; using approximate index in AIR, $f_{vi,vj}(n, \alpha)$ has an additional part, which is the extraneous traffic off the path due to false positive results from the Bloom Filters. Let $n_k$ as number of nodes whose shortest distance to the path ($v_i$, $v_j$) is $k$. For $i \neq j$, we have

$$f_{v_i,v_j}(n,\alpha) = \sum_{k=1}^{n} n_k \alpha^k \tag{12}$$

For $i=j$, $f_{vi,\ vi}(n, \alpha)$ equals to the expected network traffic for a non-hit query message starting from node $v_i$. Let $g_n(\alpha)$ be defined the same as in (8), we have

$$g_n(\alpha) = \sum_{i=1}^{n} f_{v_i,v_i}(n,\alpha) \tag{13}$$

Analogous to the proof in Theorem 1, for $1 \leq i, j \leq n$,

$$f_{v_i,v_j}(n+1,\alpha) = f_{v_i,v_j}(n,\alpha) + \frac{1}{n}(\alpha + \alpha f_{v_i,v_j}(n,\alpha)) \tag{14}$$

$$f_{v_i,v_{n+1}}(n+1,\alpha) = \sum_{j=1}^{n} \frac{1}{n}(1 + f_{v_i,v_j}(n,\alpha)) \tag{15}$$

$$f_{v_{n+1},v_{n+1}}(n+1,\alpha) = \sum_{i=1}^{n} \frac{\alpha}{n}(1 + f_{v_i,v_i}(n,\alpha)) \tag{16}$$

Let $p_n(\alpha)$ be the sum of expected network traffic for AIR to route between any pair of the $n$ nodes,

$$p_n(\alpha) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{v_i,v_j}(n,\alpha) \tag{17}$$

Summing (14), (15) and (16) we have

$$p_{n+1}(\alpha) = p_n(\alpha) + n(2+\alpha) + \frac{2+\alpha}{n} p_n(\alpha) + \alpha + \frac{\alpha}{n} g_n(\alpha) \tag{18}$$

We know from Theorem 1 that $g_n(\alpha)/n$ converges to constant. If $p_n(\alpha)$ to grow faster than O($n$), we have

$$p_{n+1}(\alpha) \approx p_n(\alpha) + \frac{2+\alpha}{n} p_n(\alpha) \tag{19}$$

Taking derivative of (19) over $n$, we have

$$\frac{\mathrm{d}p_n(\alpha)}{\mathrm{d}n} = p_{n+1}(\alpha) - p_n(\alpha) = \frac{2+\alpha}{n} p_n(\alpha)$$

$$\Rightarrow p_n(\alpha) = O(n^{2+\alpha}) \tag{20}$$

Let $q_n(\alpha)$ be the average expected network traffic for routing a message between any pair of nodes in an evolvement tree of $n$ nodes, we have

$$q_n(\alpha) = \frac{p_n(\alpha)}{n(n-1)} = O(n^\alpha)$$ □

The exact close formula of $q_n(\alpha)$ is one of our ongoing work. However, now we know asymptotically how fast $q_n(\alpha)$ grows with $n$. Next, we ask how fast the network traffic of accurate routing grows with $n$. We know that it equals to the length of the path connecting the two nodes on the evolvement tree. Therefore, we study the average path length in the evolvement tree.

**Theorem 3.** The average path length in an evolvement tree is O(log$n$) hops.

**Proof of Theorem 3:** We observe that by taking derivative of (5) over $\alpha$, we have

$$\frac{df_{v_i}(n,\alpha)}{d\alpha} = \sum_{i=1}^{n} i n_i \alpha^{i-1} \tag{21}$$

When $\alpha$ equals 1, the right side of (21) is exactly the average length of all paths in the evolvement tree that start from node $v_i$. Let $t(n)$ be the total length of all paths in the evolvement tree, we have

$$t(n) = \frac{\mathrm{d}g_n(\alpha)}{\mathrm{d}\alpha}\Big|_{\alpha=1} \tag{22}$$

Taking derivative of recursion formula in (8), we have

$$\frac{\mathrm{d}g_{n+1}(\alpha)}{\mathrm{d}\alpha}\Big|_{\alpha=1} = \frac{\mathrm{d}(1+\frac{2\alpha}{n})g_n(\alpha)}{\mathrm{d}\alpha}\Big|_{\alpha=1} + 2\alpha \tag{23}$$

When $\alpha$ is 1, $g_n(\alpha)$ represents the network traffic of flooding the whole evolvement tree, which equals $n(n\text{-}1)$. Substituting $g_n(1)=n(n\text{-}1)$ and (22) into (23), we have

$$t(n+1) = (1+\frac{2}{n})t(n) + 2n$$

$$\frac{\mathrm{d}t(n+1)}{\mathrm{d}n} = \frac{2}{n}t(n) + 2n \Rightarrow t(n) = O(n^2 \log n) \tag{24}$$

Let $d(n)$ be the average path length in the evolvement tree,

$$d(n) = \frac{t_n(\alpha)}{n(n-1)} = O(\log n) \tag{25}$$

□

Therefore, we know that AIR routes a query message to the destination node on the evolvement tree within O(log$n$) steps, at the total cost of O($n^\alpha$) network traffic. When the query has no matching object in the system, it is given a negative answer in $2\alpha/(1\text{-}2\alpha)$ steps.