Analysis and Comparison of P2P Search Methods *

Dimitrios Tsoumakos
Department of Computer Science
University of Maryland, College Park
dtsouma@cs.umd.edu

Nick Roussopoulos
Department of Computer Science
University of Maryland, College Park
nick@cs.umd.edu

Abstract

The popularity attributed to current Peer-to-Peer applications makes the operation of these distributed systems very important for the Internet community. Efficient object discovery is the first step towards the realization of distributed resource-sharing. In this work, we present a detailed overview of existing search methods for unstructured *Peer-to-Peer networks. We analyze the performance of the algorithms relative to various metrics, giving emphasis on the success rate, bandwidth-efficiency and adaptation to dynamic network conditions. Simulation results are used to empirically evaluate the behavior of nine representative schemes under a variety of different environments.*

1 Introduction

Peer-to-Peer (hence P2P) computing represents the notion of sharing resources available at the edges of the Internet [20]. A large number of systems and architectures that utilize this technology have emerged since its initial success ([7, 9, 14], etc.). The P2P paradigm dictates a fully-distributed, cooperative network design, where nodes collectively form a system without any supervision. Its advantages include robustness in failures, extensive resource-sharing, self-organization, load balancing, data persistence, anonymity, etc.

Today, many popular P2P applications operate on *unstructured* networks. In these systems, peers connect in an ad-hoc fashion, the location of the documents is not controlled by the system and no guarantees for the success or the complexity of a search are offered. More important, peers have a local only knowledge of the network in which nodes enter and leave frequently.

In this work, we describe a variety of proposed search algorithms for unstructured P2P networks. A search process includes aspects such as the query-forwarding method, the set of nodes that receive query-related messages, the form of these messages, local processing, stored indices and their maintenance, etc. This paper describes many representative approaches and analyzes their performance. We focus on the behavior of these algorithms for each of the following metrics: Efficiency in object discovery (*accuracy* and number of *hits*), bandwidth consumption and adaptation to changing topologies and workloads. The first metric measures search *accuracy* and the number of discovered objects per request (this is very important for many applications, as it enables efficient object retrieval). Minimizing message production always represents a high-priority goal for all distributed systems. Finally, it is important that any search algorithm adapts to changing conditions, since in most P2P networks users frequently enter and leave the system, as well as update their collections.

To evaluate our analysis, we simulate nine of the described methods and present a direct quantitative comparison of their performance. We identify the relative advantages and disadvantages of each method as well as the conditions under which they can be most or least effective. To our knowledge, this is the first work that attempts a direct comparison of a large and diverse set of search techniques proposed for unstructured P2P systems. We believe this is an important contribution that can provide a better understanding of the various search mechanisms and assist in choosing an algorithm that best fits a particular application.

2 Related Work

Gnutella [7] and Napster [13] are the focus of two measurement studies: Reference [18] attempts a detailed characterization of the participating end-hosts, while [2] measures the locality of stored and transferred documents. In [19], a traffic measurement for three popular P2P networks is being conducted at the border routers of a large ISP. Extensive results for traffic attributed to HTTP, Akamai and P2P systems are also presented in [17]. Quantitative comparisons between the search methods in [10, 11] and the original Gnutella algorithm are presented in these two papers. Their main comparison metric is bandwidth consumption. The work in [23] presents a thorough comparison be-

*This material is based upon work supported by, or in part by, the U.S. Army Research Laboratory and the U.S. Army Research Office under contract/grant number DAAD19-01-1-0494

tween the proposed algorithm and the search schemes introduced in [5, 11] on a variety of metrics.

Our work focuses exclusively on proposed search methods for unstructured P2P networks and provides a direct experimental comparison of these algorithms under different environments.

3 P2P Search Algorithms

We first describe our system model for search in unstructured P2P networks. Peers and documents (or objects) are assumed to have unique identifiers, with object IDs used to specify the query target. Nodes that are directly linked in the overlay are called *neighbors*. We assume that peers obtain only a local knowledge of the network (e.g., are only aware of their neighbors). Each peer retains a collection of documents, while it makes requests for those it wishes to obtain. Search algorithms cannot in any way dictate object placement and replication in the system.

Objects are assumed to be of varying popularity, which affects the respective number of replicas and received requests. The *replication distribution* dictates the number and identity of objects stored at each node. The *query distribution* controls the frequency of requests for each object.

A search is *successful* if it discovers at least one replica of the requested object. The ratio of successful to total searches made is called the *success rate* (or *accuracy*). A search can result to multiple discoveries (or *hits*), which are replicas of the same object stored at different nodes. A global *TTL* parameter represents the maximum hop-distance a query can reach before it gets discarded.

3.1 Search Taxonomy

There are two general strategies used to search for an object: Search in a *blind* fashion, trying to propagate the query to a sufficient number of nodes in order to satisfy the request; or utilize information about document locations and thus perform an *informed* search. The semantics of the used information range from simple forwarding hints to exact object locations. The placement of this information can also vary: In centralized approaches (e.g., [13]), a central directory known to all peers exists. Distributed approaches can also be sub-divided into *pure* and *hybrid*. In *pure* approaches (e.g., [4, 10, 23]), all participating peers maintain some portion of the information. In *hybrid* architectures (e.g., [5]), certain nodes assume the role of a *super-peer* and the rest become *leaf-nodes*. Each super-peer acts as a proxy for its leaf-nodes by indexing all their documents and serving their requests.

The semantics of the stored indices in informed approaches can be used for another categorization. Indices might relate to exact object locations (e.g., [21]), probability of discovery through a link (e.g., [23]), number of objects through a link (e.g., [4]), or other metrics (e.g., [1]).

Finally, we can categorize search schemes according to the query forwarding method into *flood-based* (utilizing the standard flooding scheme or one of its variations) and non *flood-based* (e.g., [11, 15]).

3.2 Blind Search Methods

GNUTELLA [7]: The original Gnutella algorithm (or *flooding* scheme) contacts all accessible nodes within TTL hops. Its basic characteristics are its simplicity and the huge overhead it produces by contacting many nodes.

Modified-BFS [10]: In this variation of the flooding scheme, peers randomly choose only a ratio of their neighbors to forward the query to. This reduces the average message production, but still contacts a large number of peers.

Iterative Deepening: Two similar approaches that use consecutive BFS searches at increasing depths are described in [11, 24]. These algorithms achieve best results when the search termination condition relates to a user-defined number of hits and it is possible that searching at small depths will satisfy the query. In a different case, they produce even bigger loads than the standard flooding mechanism.

Random Walks [11]: In *Random Walks*, the requesting node sends out k query messages to an equal number of randomly chosen neighbors. Each of these messages follows its own path, having intermediate nodes forward it to a randomly chosen neighbor at each step. These queries are also known as *walkers*. A walker terminates either with a success or a failure. Failure can be determined by two different methods: The TTL-based method and the *checking* method, where walkers periodically contact the query source asking whether the termination conditions have been satisfied. The algorithm achieves a message reduction by over an order of magnitude compared to the standard flooding scheme. It also achieves some local load balancing, since no nodes are favored in the forwarding process over others. On the other hand, success rates and number of hits vary greatly depending on network topology and the random choices made. Another drawback of this method is its inability to adapt to different query loads, since queries for popular and unpopular objects are treated in the same manner.

3.3 Informed Search Methods

Super-Peer approaches: In Gnutella2 (G2) [22], when a super-peer (or *hub*) receives a query from a leaf, it forwards it to its relevant leaves and also to its neighboring hubs. These hubs process the query locally and forward it to their relevant leaves. No other nodes are visited with this algorithm. Neighboring hubs regularly exchange local repository tables to filter out unnecessary traffic. In GUESS [5], a search is conducted by iteratively contacting different (not necessarily neighboring) super-peers and having them ask all their leaf-nodes, until a number of objects are found. The order with which super-peers are chosen is not specified.

Both schemes rely on a dynamic hierarchical structure of the network. They present similar solutions for reducing the effects of flooding by utilizing the structure of hybrid networks. The number of leaf-nodes per super-peer must be kept high, even after node arrivals/departures. This is the most important condition in order to reduce message forwarding and increase the number of discovered objects.

Intelligent-BFS [10]: This is an informed version of *modified-BFS*. Nodes store query-neighborID tuples for recently answered requests from (or through) their neighbors in order to rank them. First, a peer identifies all queries similar to the current one, according to a query similarity metric; it then chooses to forward to a set number of its neighbors that have returned the most results for these queries. If a hit occurs, the query takes the reverse path to the requester and updates local indices. This approach focuses more on object discovery than message reduction. At the cost of an increased message production compared to *modified-BFS* (because of the update process), the algorithm increases the number of hits. It achieves high accuracy, enables knowledge sharing and induces no overhead during node arrivals/departures. On the other hand, its message production is very large and only increases with time as knowledge is spread over the nodes. It shows no easy adaptation to object deletions or peer departures, because the algorithm does not utilize negative feedback and forwarding is based on ranking. Finally, its accuracy depends highly on the assumption that nodes specialize in certain documents.

APS [23]: In *APS*, each node keeps a local index consisting of one entry for each object it has requested per neighbor. The value of this entry reflects the relative probability of this node's neighbor to be chosen as the next hop in a future request for the specific object. Searching is based on the deployment of k walkers and probabilistic forwarding. Each intermediate node forwards the query to one of its neighbors with probability given by its local index. Index values are increased on successful paths and decreased otherwise. The quality of the *APS* indices is refined as more queries are made. Every node on the walkers' paths updates its indices according to search results, so peers eventually share and adjust this knowledge with time. Walkers are directed towards objects or gradually redirected if misses occur. *APS* is also very bandwidth-efficient (achieving very similar levels with *Random Walks*) and induces zero overhead over the network at join/leave/update operations. The *s-APS* modification adaptively selects the most bandwidth-efficient strategy to minimize the amount of update massages along the reverse path. On the other hand, these advantages are mainly seen when many different peers contribute with their workloads. Moreover, both *APS* and *Random Walks* have k as an upper bound in their hits per search.

Local Indices (LI) [24]: Each node indexes the objects stored at all nodes within a certain radius r and can answer queries on behalf of all of them. A search is performed in a BFS-like manner, but only nodes accessible from the requester at certain depths process the query. To minimize the overhead, the hop-distance between two consecutive depths must be $2r+1$. This approach resembles the two search schemes for hybrid networks. The method's accuracy and hits are very high, due to the indexing scheme. On the other hand, message production is comparable to flooding, even if the processing time is smaller because many nodes just forward the query. The scheme requires a flood with $TTL = r$ whenever a node joins/leaves the network or updates its local repository, so the overhead becomes even larger for dynamic environments.

GIA [1]: In *GIA*, requesting nodes deploy biased walkers in order to discover various objects. Each peer chooses to forward the query to the neighbor with the highest announced *capacity*. This is a user-defined metric that reflects the processing power of a node inside the system. Moreover, the protocol requires that each peer indexes the documents of its neighbors. This scheme also utilizes a topology-adaptation algorithm which re-configures the overlay connectivity such that each node is connected to a number of peers proportional to its capacity. The biased walkers are then directed towards highly connected neighbors and, probabilistically, to those with the highest number of indexed objects. Finally, the scheme provides a flow-control mechanism which allows peers to control the rate at which they can accept and process requests from their neighbors. Once the topology has been set, we expect *GIA* to perform very bandwidth-efficient searches with several hits. On the other hand, the adaptation algorithm plus the indexing of the neighbors' repositories increase the responsibilities of each peer as well as the communication overhead. Another issue is how fast can the algorithm work for joining peers and at what cost for their neighborhood.

Routing Indices (RI) [4]: Documents are assumed to fall into a number of thematic categories. Each node stores an approximate number of documents from every category that can be retrieved through each outgoing link (i.e., not only from that neighbor but from all nodes accessible from it). The forwarding process is similar to DFS: A node that cannot satisfy the query stop condition with its local repository will forward it to the neighbor with the highest "goodness" value. Three different functions which rank the out-links according to the expected number of documents discovered through them are also defined. The algorithm backtracks if more results are needed. This approach trades index maintenance overhead for increased accuracy. While a search is very bandwidth-efficient, RIs require flooding in order to be created and updated, so the method is not suitable for highly dynamic networks. Moreover, stored indices can be inaccurate due to thematic correlations, over- or under-counts in

document partitioning and network cycles.

In [15], each node holds a number of *bloom* filters for each neighbor. The i^{th} filter summarizes documents that can be found i hops away through that specific link. Nodes forward queries to the neighbor whose smaller depth bloom filter matches a hashed representation of the object ID. After a certain number of steps, if the search is unsuccessful, it is handled by a deterministic algorithm instead of backtracking. The scheme's expectation is to find only one replica of the object with high probability. Index maintenance requires flooding messages initiated from nodes that arrive or update their collections.

Distributed Resource Location Protocol (DRLP) [12]: Nodes with no information about the location of a document forward the query to each of their neighbors with a certain probability. If an object is found, the query takes the reverse path to the requester, storing the document location at those nodes. In subsequent requests, nodes with indexed location information directly contact the specific node. If that node does not currently obtain the document, it just initiates a new search as described before. This algorithm initially utilizes flooding to find the locations of an object. In subsequent requests, it might take only one message to discover it. A small message production is achieved only with a large workload that enables the initial cost to be amortized over many searches. In rapidly changing networks, this approach fails and more nodes have to perform blind search. This also affects the number of hits: If many blind searches are made, then many results are found; if many direct queries take place, then only one replica is discovered.

Gnutella with Shortcuts (GS) [21]: In this work, the authors propose the addition of *shortcuts* (i.e., direct links to peers that have recently proved useful in answering queries) to a Gnutella-like overlay. The original flooding mechanism is initially used to locate documents. Peers that provide answers are indexed by the requesters, following the assumption that they could provide answers to more requests. When a new query is made, nodes first forward it to their shortcuts (ranked in a descending order of usefulness — usually the success rates). If all shortcuts fail, the standard flooding scheme is again used to locate the object. This approach resembles the *DRLP* scheme but stores more than one pointer and keeps statistics on them. For semantically related queries, we expect it to quickly identify relevant peers and mostly use the shortcuts for object location. Moreover, we anticipate a very high success rate since the fall-back mechanism is flooding. On the other hand, if peers make many unrelated queries or they do not store relevant content, it is possible that the shortcuts will fail, which in turn means that the system pays the price of a full-scale flooding. The same is true when objects are removed or peers depart frequently.

New Approaches: Recently, there has been an effort to combine the advantages of structured systems (also known as Distributed Hash Tables — DHTs) and unstructured ones. In [6], an *immediate neighborhood* area is defined for each peer. Object placement inside these overlapping areas is performed in a DHT-like fashion. Searches use the standard flooding mechanism except that only certain areas are probed. In [3], peers are grouped into *possession rules*, according to whether they contain a specific item or not. Nodes search inside one possession rule in a blind fashion. The possession rule is chosen by a greedy mechanism according to past query results.

4 Simulation Results

In this section we present results for nine of the described methods: (*G2, Random Walks, Modified-BFS, Intelligent-BFS, Local Indices, s-APS, DRLP, GS* and *GIA*). The simulated methods are representative blind and informed schemes, both flood and non flood-based, with or without user-initiated index updates (that is, updates triggered strictly by the search process). We briefly summarize our simulation model here.

In our experiments, we utilize the GT-ITM [25] and Inet-3.0 [8] topology generators to produce sets of random and power-law graphs respectively. We assume a pure P2P model, where all peers equally make and forward requests. For each setup, the results are averaged over a set of 10 similar graphs for each described topology. We also present results on a real gnutella graph [16], with 61,685 nodes and average degree $d = 4.6$.

By default, we assume 100 objects of decreasing popularity, with object 1 being the most popular. The small number of objects enables a better observation of the effect that popularity has over the performance of a search. A zipfian distribution with parameter $a = 0.82$ is used to model both query and replication distributions and achieve workloads similar to [2]: The top-10% of the objects account for about 50% of all stored objects and receive about 50% of all requests. The replication ratios range from 11% to 0.25% for the least popular object. With this distribution, around 8,500 replicas of all 100 objects exist inside our network (each peer holds less than one object on average). Requester nodes are randomly chosen and represent about 20% of the total number of nodes. Each requester makes about 1,500 queries over a time period. We do not allow extra replicas to be stored (i.e., we only consider the search phase, not object retrieval). Finally, the TTL parameter is set to 5, since larger values produced very similar results.

To simulate dynamic network behavior, we insert offline nodes and remove active ones with varying frequency. The objects are also re-distributed to model file insertions and deletions. We always keep approximately 80% of the network nodes active, while arriving nodes start functioning without utilizing any (possibly built) prior knowledge.

Object re-location always follows the initial distribution parameters.

The *Intelligent-BFS* method was modified to allow for object-ID requests. Index values at peers now represent the number of replies for an object through each neighbor and nodes choose the neighbors with the highest index values when forwarding a query. For *Modified-BFS*'s, *DRLP*'s and *Intelligent-BFS*'s flood-based search, nodes choose an equal number of neighbors to forward a query in order to make direct comparisons. For *G2/GUESS*, peers randomly choose k neighbors to forward the query to. The chosen nodes forward the query to all their neighbors. By modifying the value for k we can simulate the operation of both *G2* (with k always larger than the average node degree) and *GUESS*. In our simulations, *G2/GUESS* operate on a pure (instead of a hybrid) model in order to achieve uniformity in our results. Moreover, they both function in a blind manner, so no cache or repository table exchange takes place. We name this approach *HG2* (Hybrid G2/Guess). For our *LI* implementation, nodes index the objects of their neighbors ($r = 1$). To ensure that the search is equivalent to a flood with $TTL = 5$, only peers at depths 1 and 4 process the query. We also ensure that no object from the same peer is being discovered multiple times. Finally, our *GIA* implementation deploys k walkers, with each peer forwarding to the neighbor with the highest out-degree, while the overlay adaptation process is not simulated. Peers index the documents of their immediate neighbors. For our *GS* implementation, we use 5 shortcuts and rank them by their success rates.

4.1 Basic performance comparison

In our first set of experiments, we use a set of 10,000-Node random graphs (average degree $d = 4$) to compare the nine methods over 5 different environments: A static one, one with low/high object relocation frequency and one with low/high peer departure frequency. In the two low-frequency scenarios, relocation and departures/arrivals occur about 300 times per run, while in the high-frequency ones they occur 10 times more often. *DRLP* and *Int/Mod-BFS* forward to 3 neighbors, while $k = 7$ for *s-APS, GIA, HG2* and *Random Walks*. Figures 1 and 2 present the results.

Blind methods show a fairly stable performance between the static and dynamic settings, since the dynamic operations do not interfere with the forwarding scheme. Flood-based schemes discover many objects at a higher cost. Nevertheless, only *LI* and *GS* with the pure-flooding scheme achieve very high accuracy. This happens because of the small out-degree of our network. We also notice that blind and flood-based techniques do not get affected by object relocation, but only by peer joins/leaves. While our relocation process does not substantially alter anything in those algorithms' operation, peer arrivals/departures alter the topology and the amount of available resources.

Mod/Int-BFS show relatively high accuracy and return many hits. Their performance is very similar, with the informed method showing marginally better results. For environments resembling this setup, *Mod-BFS* will be preferred, since its performance is equally high and it is much simpler. We expect the informed method to perform better in richer or more specialized environments (like the one described in [10]), mainly in the number of hits.

Random Walks displays low accuracy (<34%) and finds less than 0.5 objects on average. Its bandwidth consumption is quite low (about 15 messages), while its performance is hardly affected by the dynamic operations. *HG2* behaves similarly, with the exception of producing about 5 more messages per search. In general, these algorithms exhibit poor performance and appear very robust to increased network variability. This is reasonable, as walkers are randomly directed with no regard to topology or previous results.

The *s-APS* method achieves a success rate of over 75% in the static run, a number that drops by around 30% in the highly dynamic settings, but only around 12% in the two less dynamic ones. The metric that is reasonably affected is the number of discovered objects, which are almost cut to a third. This happens because it takes some time for the learning feature to adapt to the new topology and paths to discovered objects frequently "disappear". On the other hand, it manages to keep its messages almost as low as *Random Walks*'. The scheme is equally affected by relocations and departures/arrivals, since walkers are directed towards specific locations which are altered by both types of events. Nevertheless, it exhibits a good overall performance compared to the non-BFS related schemes, without indexing other peers' repositories.

The *DRLP* algorithm exhibits some interesting characteristics. First, its message production is very low (less than 6 messages per request). Our simulations count the direct contact of a node (both for *DRLP* and *GS*) as one message, although a link between them might not exist in the overlay. Dynamic behavior causes the stored addresses to become more frequently "stale", thus the initial flooding is performed more often. This is the reason for the decrease in its accuracy from 99% in the static case to 77% and 15% in the highly dynamic ones. *DRLP* produces the same amount of messages for its initial search with *Modified-BFS*, so it needs many successful requests to amortize this initial cost. The number of objects it discovers is very small, ranging from 1.4 to 0.2. If *DRLP* is forced to use flooding many times, then the number of hits increases. If it is successful and produces few messages, then it only finds one replica per request. Despite this, we notice that it proves very bandwidth-efficient and flooding is scarcely used. This is due to the fact that, with many nodes making requests, most

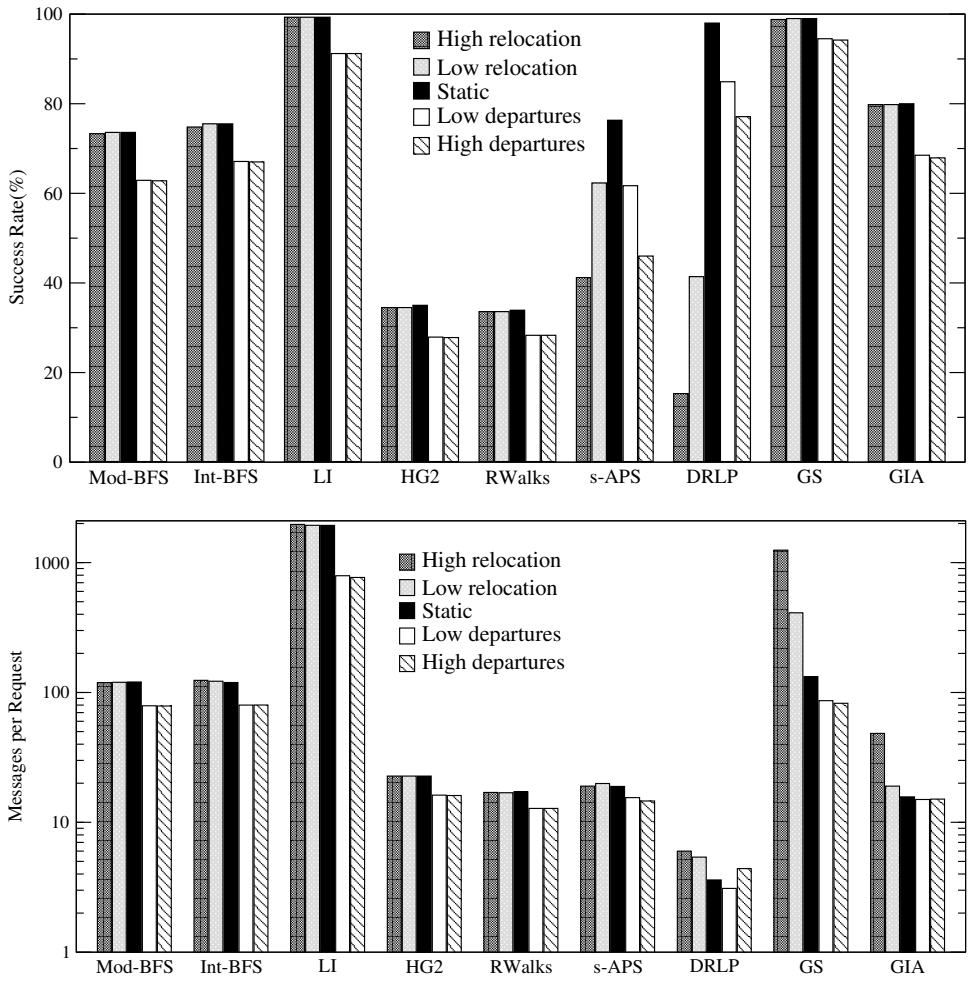

Figure 1. Success rate and message production of the methods using a set of 10,000-node random graphs with average degree $d = 4$

of them obtain a pointer for every object after a while. So, even if some node initiates a flood, most of its neighbors will only forward to one other node. The large number of requests per run helps *DRLP* achieve a very low average message consumption. This scheme seems ideal for relatively static environments and large workloads, with the exception that the number of hits will be very close to one. Another important observation is that *DRLP* is affected far more by object relocation than by node departures. This is reasonable if we consider that with departures there still exist nodes with a valid pointer to an object, whereas object relocation may make many pointers become stale at once.

The *LI* scheme proves the most productive in terms of discovered locations and the most costly in message production. It produces one order of magnitude more messages than the other BFS-related methods but also discovers about 10-20 times more objects, taking advantage of its index scheme. Its performance is only affected by the dynamic joins and leaves, with a decrease of more than 50% in located objects. The cost of the index updates, even under the more dynamic settings, is negligible compared to the cost of a search (at most 2% over the total number of messages). On the other hand, this cost is considerable for nodes that stay idle (and possibly alter their local repositories), since it induces traffic without any search involved.

GS shows very high accuracy, since it can always fall back to the flooding scheme. Nevertheless, when peers do not have shortcuts or when these fail (this happens mostly when objects get relocated), message consumption increases dramatically. On the other hand, similarly to *DRLP*, the more flood searches are performed, the more objects are discovered. Shortcuts are mostly used in the static and dynamic arrival/departure modes, since 5 shortcuts proved sufficient for at least one of them to provide an answer most of the times.

Finally, *GIA* manages to perform as well as *Mod/Int-BFS* but being more bandwidth-efficient. The combination of one hop indexing and biased walkers achieves a good, ro-

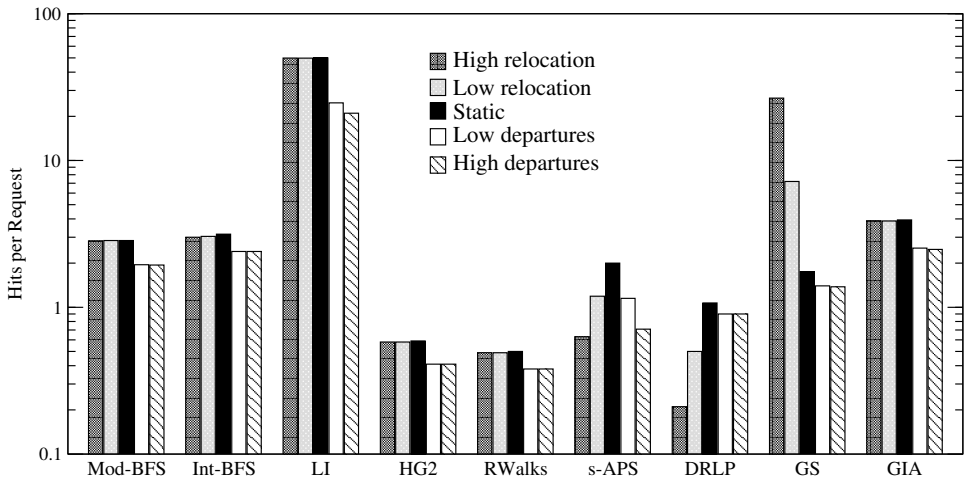

Figure 2. Hits per query of the methods using the set of 10,000-node random graphs with average degree $d = 4$

bust performance at relatively low cost. Only in the high relocation setting we notice a considerable increase (200%) in the average message consumption since peers have to refresh their indices frequently.

4.2 Results on more dense graphs

In the next set of simulations we use a random graph set with an average degree $d = 10$ to compare the 9 methods over two different environments: A static one, and one where both object relocation and peer departures occur about 600 times per run. *DRLP* and *Int/Mod-BFS* forward to 4 neighbors at each step, while $k = 12$ for *s-APS, Random Walks, HG2, GIA*. All other parameters remain the same. The results for the static case are shown in Table 1. We also report the percentage of messages per search that are duplicates and the average distance of the hits in overlay hops.

Blind forwarding causes a large amount of messages to be dropped. Informed methods with no direct indices perform much better (*s-APS, Int-BFS* wasting only 0.1% and 0.4% of their messages respectively). Flood-based schemes also exhibit large hop distances for their hits.

All algorithms produce a larger number of messages per request in the new graph, taking advantage of the larger number of connections. *DRLP* still manages to average less than 10 messages per request. *Random Walks* and *s-APS* roughly double their hits and increase their accuracy. On the other hand, *Int/Mod-BFS* produce 10 times more messages. *HG2* performs in between, producing about 5 times more messages. *LI* increases its bandwidth production by more than an order of magnitude. The overhead due to update messages is even less apparent now, since its search messages overshadow their effect. *GS*'s performance increases similarly to *LI*'s since they use the same underlying mechanism. Finally, *GIA* exhibits a very good performance again, having low message consumption and increased accuracy/hits.

Another interesting metric is the percentage of hits discovered at various distances by the methods (Figure 3). It shows how many objects each method locates with few or more messages. Our discussion is based on the static setting. Flood-based schemes discover the vast majority of the objects TTL hops away, since the available nodes increase exponentially with distance. *LI* always locates about 99% of its objects 4 hops away, and the rest only 1 hop away from the requesters (since only nodes at these two depths process the queries), while *HG2* discovers about 90% of the objects with its flooding phase (2 hops away). *Random Walks* discovers almost the same number of objects per distance, since the query forwarding is done randomly. *GIA* also uses walkers and exhibits a similar behavior as requesters are randomly chosen in our simulations. *DRLP* finds almost 70% of its hits using its indices (which also explains why its hit average is close to one). *s-APS* displays a symmetric curve. After a certain distance, possible paths become too many and the accuracy of the indices drops. Finally, we notice that *GS* only discovers about 5% of its hits using the shortcuts, whereas in the smaller graph the respective number was 50%. This can be explained by the fact that the flooding scheme now finds 2 orders of magnitude more objects than in the previous graph, while shortcuts still find one object.

Figure 4 shows how object popularity affects the methods' accuracy and message production in the dynamic environment. Popularity decreases as we move to the right along the x-axis. The first data point represents the accuracy/messages of the methods for the top-10%, the second for objects ranked between 11–20%, etc. This is an important comparison, because different applications or users target objects of varying popularity.

Table 1. Comparison on 10,000-node random graphs with degree $d = 10$

Metric	*ModBFS*	*IntBFS*	*LI*	*HG2*	*RWALKS*	*s-APS*	*DRLP*	*GS*	*GIA*
Success(%)	98.8	99.8	100	70.2	53.4	91.7	100	100	97.0
Messages	875	1233	39710	108.7	43.6	43.0	8.0	2344	35.0
Duplicates(%)	10.3	0.4	18.7	8.3	0.2	0.1	1.8	17.8	0.9
Hits	20.2	32.6	300.0	2.9	1.2	6.1	1.4	18.9	9.5
Hit Distance	4.58	4.61	3.99	1.88	2.78	3.16	1.90	4.60	3.1

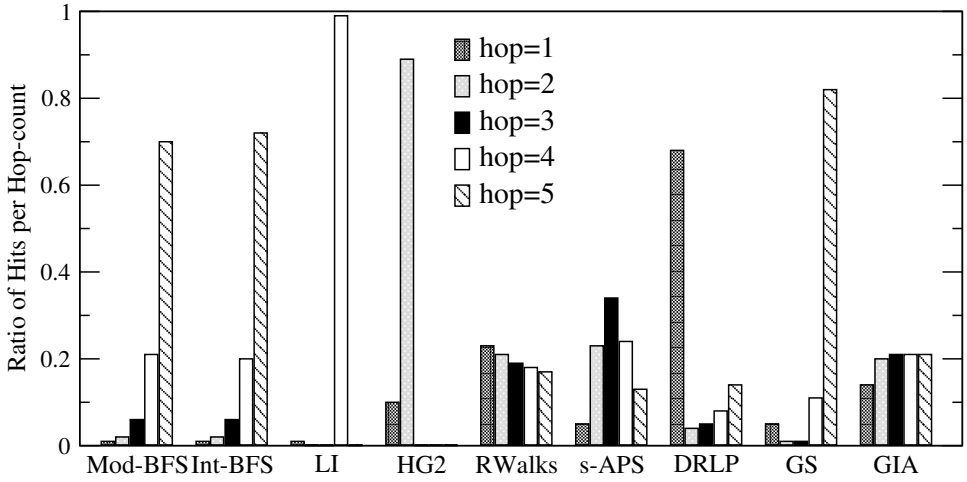

Figure 3. Hits per hop distance from the requesters

The three BFS-related methods together with *GS* exhibit very high accuracy, with *Mod-BFS* showing a noticeable decrease only for the least popular items. *Random Walks, HG2, s-APS* and *GIA* show decreasing accuracy as popularity drops, with *GIA* and *s-APS* clearly performing better. *DRLP* performs very poorly for the very popular documents (about 20%), but its accuracy increases as popularity drops. This can be explained by the fact that less popular objects receive considerably fewer queries. Therefore, object relocations and node departures which affect the algorithm happen less frequently during requests for such objects. All algorithms — except *DRLP* and *GS* — waste roughly the same amount of messages per request for each popularity group. *DRLP* and *GS* increase their consumption with a popularity decrease for the sole reason that the cost of the initial floods is now amortized over a smaller number of requests. Finally, we noticed that all algorithms (except *DRLP* and *GIA* that deploy full flooding) discover a decreasing number of objects as popularity drops, exactly because this means there exist fewer objects to be located.

In the dynamic environment, we also measure the percentage of messages per request sent due to index updates (for relevant methods only). We found that *Int-BFS* requires 11%(=131 mesg) of its messages for index updates. The respective numbers for *LI, GIA* and *s-APS* are 14.4%(= 2968 mesg), 31.7%(= 14 mesg) and 18.5%(= 8 mesg). Although *GIA* and *s-APS* appear to require a larger portion of updates, they are much more bandwidth-efficient than the other methods in absolute performance.

Our previous simulations depicted the relative performance characteristics of the nine algorithms. To some extent, that sort of comparison was not direct either because of the different nature of the methods or because of the single choice of the various parameters. Since it is impossible to directly compare the methods for the same parameter values (e.g., k, TTL), we select simulations on a third set of 10,000-node random graphs ($d = 20$), where the algorithms had similar performance in one of two important metrics: Messages and hits per query. These results were obtained by experimenting on various values for k, TTL, number of neighbors to forward and number of requester nodes. The results are presented in Table 2 and the comparison metric is typed in boldface. *LI* is omitted from this table because its large message and hit production could not be matched by the other methods.

For similar message consumption, first *GIA*, then *s-APS* discover the most objects (followed by *DRLP* with about 10 extra messages per search). These three methods also prove extremely accurate, while the rest of the schemes (either flood-based or random) do not perform well. For similar hits per search, again *GIA* and *s-APS* stand out above *DRLP*, which wastes a few more messages but is perfectly accurate. From the rest of the methods, only *GS* is 100% successful, but exhibits the highest message consumption.

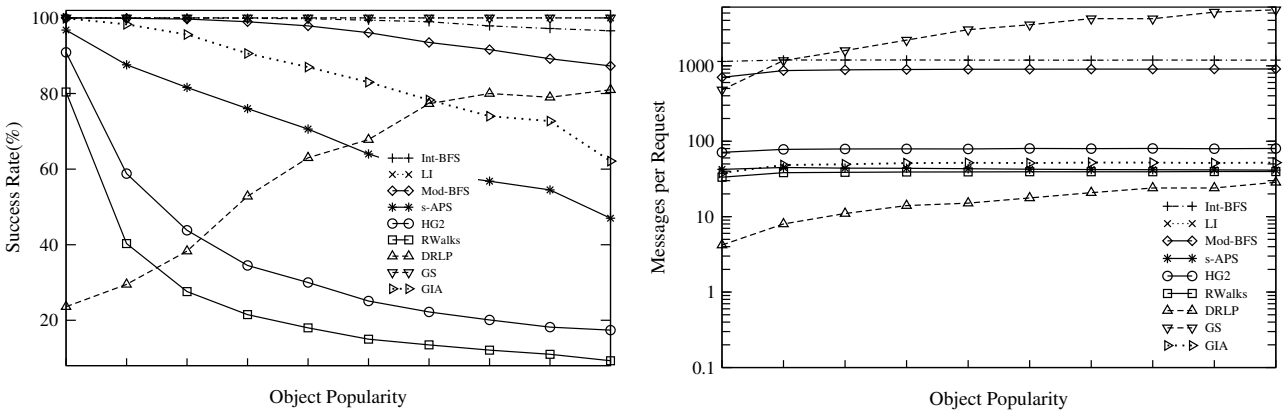

Figure 4. Accuracy and message production vs. object popularity in the dynamic setting

Table 2. Comparison on 10,000-node random graphs with degree $d = 20$

Metric		*Mod-BFS*	*Int-BFS*	*HG2*	*RWALKS*	*s-APS*	*DRLP*	*GS*	*GIA*
Messages	Success(%)	63.6	67.6	63.5	62.2	93.4	100	90.8	99.9
	Messages	**73.4**	**83.0**	**77.0**	**72.5**	**70.6**	**79.2**	**77.0**	**70.0**
	Hits	1.9	2.3	2.1	2.0	10.7	5.3	1.12	14.9
Hits	Success(%)	75.8	77.0	71.9	75.0	80.2	100.0	100.0	92.2
	Messages	134.4	117.1	115.1	125.2	31.4	43.0	356.5	32.1
	Hits	**3.5**	**3.2**	**3.1**	**3.2**	**3.8**	**3.4**	**3.6**	**3.8**

4.3 Increased number of objects

Our previous model was mainly tailored for a system where peers continuously search for specific objects. The wide range of replication ratios together with the network dynamics best enables us to observe the effect of popularity, dynamic behavior and forwarding scheme. We now consider a more general situation, with a large number of objects (20,000) and 5,000 requester nodes, each making 2,000 queries. This could be an example of a P2P search engine application, with users having their own preferences (changing with time). Table 3 presents our comparison using three sets of graphs, our original 10,000-node set ($d = 4$), a 10,000-node power-law graph set ($d = 4.4$) and a Gnutella topology snapshot ($d = 4.6$). For larger graphs (simulations up to 50,000 nodes), results are qualitatively similar.

Compared to the previous results, we clearly notice a small performance degradation, which is natural if we consider that now more queries are made for sparsely located objects, while flooding is used more by some of the methods. Nevertheless, first *DRLP*, followed by *s-APS* and *GIA* achieve numbers closest to the original ones. With the power-law topology, although the average out-degree is the same as with the random graphs, various neighborhoods differ substantially, since there are few nodes with very high connectivity. *GIA* clearly takes advantage of this to increase its discovered objects. Another observation is that pure flood-based schemes also discover substantially more objects (compared to the respective runs over the random topologies with 20,000 objects). *HG2* achieves more than 10 times more hits with a 150% increase in accuracy, using 30 times more messages. *LI* doubles its hits without any message increase. The rest of the schemes perform very similarly to the previous simulation. The results for the real topology resemble those for the power-law graphs if we also take into account the size increase as well as an increase in the average out-degree and the number of poorly connected neighborhoods (possibly due to crawling imperfections). In general, most methods show increased messages and hits compared to the random topologies. While they effectively locate popular objects, they either fail to be as accurate or greatly increase their message production for the bulk of the non-popular items.

5 Conclusions

This paper presents a description of current search techniques for unstructured P2P networks, along with a quantitative comparison through simulation. Our analyses focus on the performance metrics of search accuracy, bandwidth consumption, discovered objects and behavior under dynamic operations.

The specifics of the problem play a big role in choosing the appropriate method. Each scheme has its own goals and it is important that these goals match the application's. Important parameters that could influence our decision include the primary purpose of the application (e.g., fast discovery,

Table 3. Comparison of the nine methods with a 20,000-object pool

Graph		*Mod-BFS*	*Int-BFS*	*LI*	*HG2*	*RWALKS*	*s-APS*	*DRLP*	*GS*	*GIA*
RANDOM	Success(%)	68.4	69.7	89.9	30.7	29.8	75.2	99.0	89.2	74.4
	Messages	118.8	115.4	1511.6	24.9	18.6	24.1	7.1	563.5	18.3
	Hits	2.3	2.4	37.7	0.5	0.4	2.2	1.2	5.0	3.2
POWER-LAW	Success(%)	56.8	62.3	93.3	76.7	22.9	75.7	98.3	88.4	85.7
	Messages	73.3	82.0	1473.0	750.3	13.1	15.1	5.0	355.9	19.1
	Hits	1.5	1.8	86.1	17.7	0.3	1.9	1.2	3.0	13.9
GNUTELLA	Success(%)	67.8	76.2	94.7	63.3	33.7	70.1	99.1	83.6	78.8
	Messages	145.6	217.4	1325.1	282.1	24.3	33.1	17.1	886.5	20.9
	Hits	2.6	4.4	59.8	5.7	0.5	3.0	2.0	15.3	6.0

many hits, bandwidth-efficient and accurate, easy deployment, etc), the underlying topology, expected workload, etc. We offer some general-purpose observations based on our analysis and simulations, hoping they will prove useful in evaluating the plethora of different schemes.

a) Blind forwarding is not adequate for both high performance and low message cost. Keeping direct pointers to more peers (e.g., *DRLP, GS*) is very efficient in relatively static environments.

b) Index semantics play an important role: Direct location information is effective but sensitive to changes and more demanding (becomes obsolete if a failure/relocation occurs, requires update messages). Indirect information (e.g., success rates in *s-APS, Int-BFS* or connectivity/capacity in *GIA*) is much more robust but less accurate.

c) Indexing other peers' repositories is very useful but must be carefully applied, since it requires updates to keep the indices up-to-date.

d) Adaptation is a key characteristic through which peers that have a prolonged stay in the network enhance their knowledge with time. *GS, s-APS* and *Int-BFS* learn from system searches and improve their performance.

e) In many cases, the simple protocols are the preferred ones. The simplicity of the mechanisms behind flooding or random walks make them powerful and easy to implement. They can be used either by themselves or in combination with other schemes to improve their performance.

References

[1] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making Gnutella-like P2P Systems Scalable. In *SIGCOMM*, 2003.

[2] J. Chu, K. Labonte, and B. Levine. Availability and Locality Measurements of Peer-to-Peer File Systems. In *SPIE*, 2002.

[3] E. Cohen, A. Fiat, and H. Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. In *INFOCOM*, 2003.

[4] A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In *ICDCS*, July 2002.

[5] S. Daswani and A. Fisk. Gnutella UDP Extension for Scalable Searches (GUESS) v0.1.

[6] P. Ganesan, Q. Sun, and H. Garcia-Molina. YAPPERS: A peer-to-peer lookup service over arbitrary topology. In *INFOCOM*, 2003.

[7] http://www.gnutella.com. Gnutella website.

[8] C. Jin, Q. Chen, and S. Jamin. Inet: Internet Topology Generator. Technical Report CSE-TR443-00, Department of EECS, University of Michigan, 2000.

[9] http://www.jxta.org. Project JXTA.

[10] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A Local Search Mechanism for Peer-to-Peer Networks. In *CIKM*, 2002.

[11] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *ICS*, 2002.

[12] D. Menascé and L. Kanchanapalli. Probabilistic Scalable P2P Resource Location Services. *SIGMETRICS Perf. Eval. Review*, 2002.

[13] http://www.napster.com. Napster website.

[14] http://www.microsoft.com/net. Microsoft .NET.

[15] S. Rhea and J. Kubiatowicz. Probabilistic Location and Routing. In *INFOCOM*, 2002.

[16] M. Ripeanu and I. Foster. Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems. In *IPTPS*, 2002.

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

[18] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. Technical Report UW-CSE-01-06-02, Un. of Washington, 2001.

[19] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. In *SIGCOMM Internet Measurement Workshop*, 2002.

[20] C. Shirky. What Is P2P...And What Isn't. *OpenP2P.com*, 2000.

[21] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems. In *INFOCOM*, 2003.

[22] M. Stokes. Gnutella2 Specifications Part One: http://www.gnutella2.com/gnutella2_search.htm.

[23] D. Tsoumakos and N. Roussopoulos. Adaptive Probabilistic Search for Peer-to-Peer Networks. In *3rd IEEE Intl Conference on P2P Computing*, 2003.

[24] B. Yang and H. Garcia-Molina. Improving Search in Peer-to-Peer Networks. In *ICDCS*, 2002.

[25] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Infocom*, 1996.